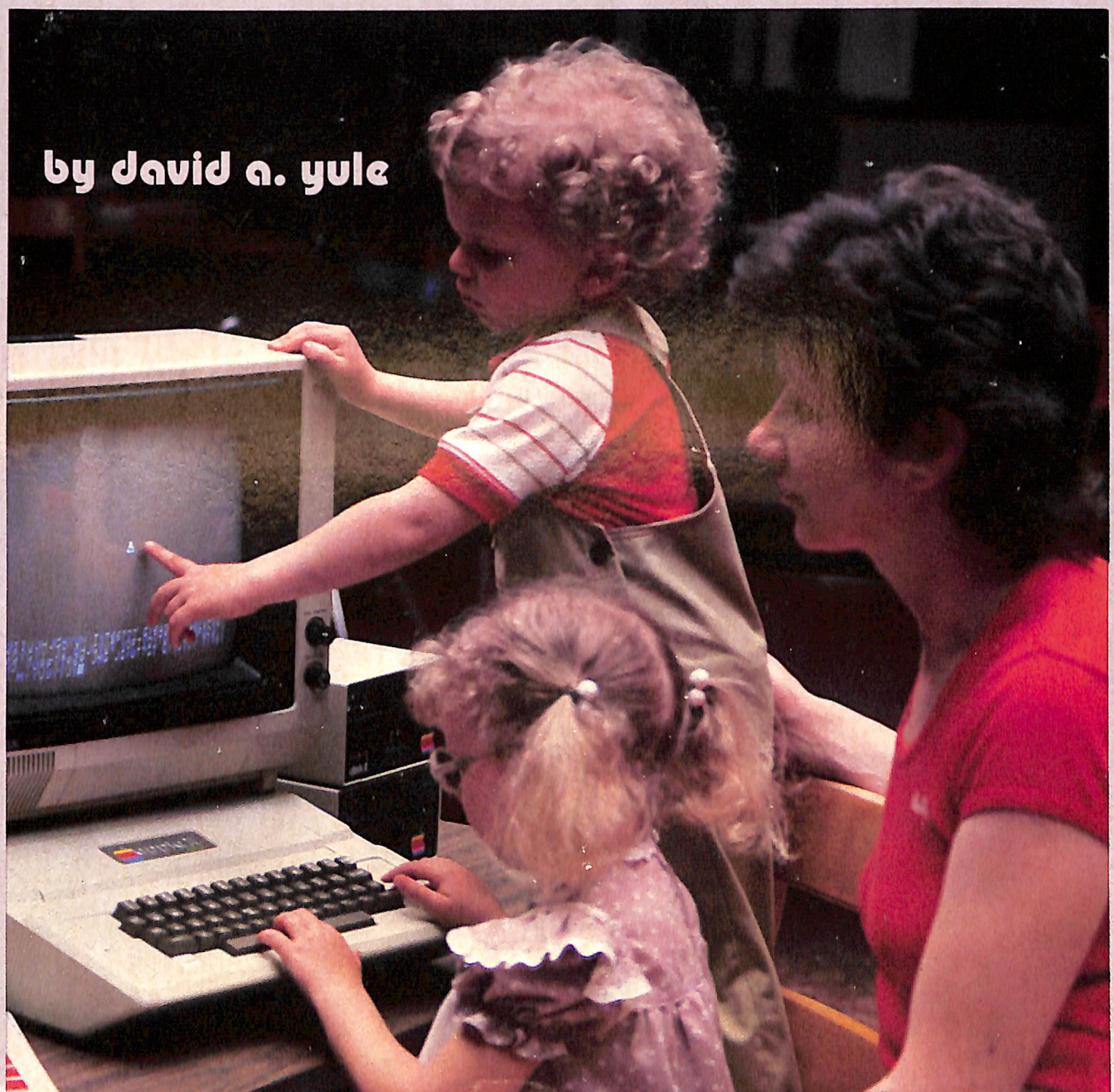
1728

apple logo for kids



by david a. yule

No. 1728
$16.95

apple logo for kids

by david a. yule

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

FIRST PRINTING

Copyright © 1984 by TAB BOOKS Inc.
Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Library of Congress Cataloging in Publication Data

Yule, David A.
Apple logo for kids.

1. Apple computer—Programming. 2. LOGO (Computer program language) I. Title.
QA76.8.A66Y85 1984 001.64'24 83-24167
ISBN 0-8306-0728-5
ISBN 0-8306-1728-0 (pbk.)

Contents

Introduction vii
To the Parent

1 Your Very Own Turtle 1
Telling Turtle What to Do—Telling Sam to Go Forward—Poor Sam Doesn't Understand—How Far Can Sam Go?—Telling Sam to Turn Right—Telling Sam to Turn Left—Sam Can Back Up!—Summary—Challenges—Quiz—Crossword Puzzle

2 Teaching Sam How To Do Things 19
Teaching Sam How to Square—Saving Your Procedures on Disk—How to Erase Sam's Memory—Loading Procedures from Your Disk—Teaching Sam How to Triangle—Making Sam Do Things Over and Over Again—Making a Dandelion—Summary—Challenges—Quiz—Crossword Puzzle

3 What To Do If You Are Stuck 44
Common Problems That Confuse Sam—Problems That Confuse You—Playing at Being Sam—Writing Out What Sam Should Do Next—Rules That Sam Must Follow—Using a Trace Function in Logo—Summary—Challenges—Quiz—Crossword Puzzle

4 If You Don't Type Well, You Will Love This! 62
Controlling Sam's Memory and Screen—Using EDIT "SQUARE—Tutorial—Renaming a Procedure—Defining a Procedure from the Editor—Some Advanced Editing Features—Teaching Sam to Do Things with One Keystroke—Making Sam Wait—Summary—Challenges—Quiz—Crossword Puzzle

5 Using Color with Sam 83
Colors That You Can Use—Choosing the Background Color—Choosing the Pen Color—Pen Up and Pen Down—Changing Sam's Screen—Summary—Challenges—Quiz—Crossword Puzzle

6 You Like Circles, Don't You? 102
Drawing Circles by Yourself—Letting Sam Draw His Own Circles: Automatic Pilot—How Useful Is Part of a Circle?—Wiggle Waggle with Sam—Lots of Arcs—Lots of Pictures—Summary—Challenges— Quiz —Crossword Puzzle

7 Telling Sam to Draw Large and Small 122
What Is a Variable?—A Triangle of Any Size—A Dandelion of Any Size—Using Two Variables—Changing Variable Values within a Procedure—A Better Flower for Sam to Draw—A Tower of Blocks for Sam to Draw—A Visit to Egypt with Sam—How Many Variables Can Sam Handle Anyway?—Summary—Challenges—Quiz—Crossword Puzzle

8 Telling Sam, to Tell Sam, to Tell Sam . . . 145
Recursive Squares—Drawing Many Circles—Stopping a Procedure in Style—Printing a Message on the Screen—Recursion and What It Means to Sam—Polygons—Drawing Many Circles—Using the Things That We Have Learned in One Procedure—Spinning Circles with Sam—Making Smaller and Smaller Circles with Recursion—Sam's Tower Revisited—Sam's Pyramid Revisited—Using Recursion to Make a Daisy Chain—Summary—Challenges—Quiz—Crossword Puzzle

9 Drawing Pictures with Sam 172
Some Things We Know That Will Help—Planning Your Picture or Making It Up As You Go—Setting Backgrounds and Colors—Sunsets—"Stop Right There!" Said Sam—Drawing Mountains—Trees—Putting the Parts Together—Harry Mouse—Challenges—Crossword Puzzle

Appendix A A Parent's Guide to Logo 195
What Can Your Child Learn from Logo?—Briefly, How Does Logo Work?—What Can I Do to Help My Child in Using Logo?—Your Handy Dandy List of Never Fail Questions

Appendix B Getting Started with Apple Logo 199

Appendix C Getting Started with Terrapin Logo 201

Appendix D Summary of Common Apple Logo Commands 202

Appendix E Equivalent Commands and Keystrokes For Apple Logo, Terrapin Logo, TI Logo, and TRS-80 Color Logo 207

Appendix F Crossword Puzzle Solutions 209

Index 212

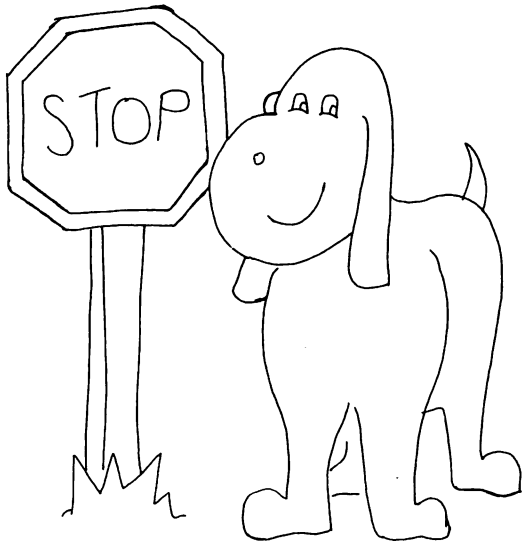
Introduction

The purpose of this book is to introduce you to the world of computers and programming through the computer language called Logo. You will find that Logo is very easy to use. It is still a complete language, however, and it has most of the structures of other well-known languages such as BASIC and PASCAL.

Which version of Logo do you have? This book is written for Apple Logo. If you have TI Logo, Terrapin Logo, or Radio Shack Color Logo, then you should look at Appendix (E). In Appendix E, you will find a table of commands, or *primitives*, and equivalent keystrokes for the different Logos available.

Here is how to use this book.

Begin at Chapter 1 and work your way through the chapters. As you move through the book, you will see some symbols that will help you. When you see a little dog that looks like this watchdog you will know that this part of the text contains something that is special. It



might be something in Logo that is tricky, is used a lot, or is especially helpful to you.

At the end of each chapter, you will find a summary of all of the new things mentioned in

that chapter. It is probably best that you read the summary carefully. After the summary, you will find a number of challenges and suggestions that you might want to try. You can use the ideas provided, or you can make up your own Logo challenges.

After the list of challenges, you will find some questions on the chapter. The questions are not intended to be difficult. All of the answers will be found in the previous chapter and summary. Most of the answers are simply words that are left out of the summary for that chapter. If you have trouble with a question, simply look back to the summary to find out what the missing word is.

Finally, at the end of each chapter, you will find a crossword puzzle. The words in the crossword puzzle are all words that are new in that chapter. The solutions to the crossword puzzles are given in Appendix F.

Chapters 3 and 4 are optional. Chapter 3 is on how to solve problems, and it has very few procedures in it. You might want to look at Chapter 3 after you have tried some of the later chapters. Chapter 4 is on editing and it is very useful, but you can manage to use the rest of the book without it. It is very convenient, however, if you know how to edit the procedures you will be designing.

Most of all, Logo is fun. You will soon feel at home with the "turtle" and will feel confident in "telling turtle" what to do.

TO THE PARENT

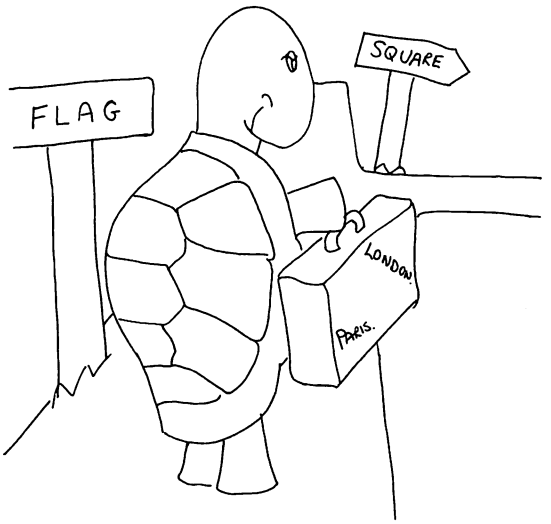
After your son or daughter reads this book, you will find that he or she should be able to write programs in Logo that will impress you with both their simplicity and their beauty. Logo can be used by a wide variety of age groups at a wide variety of levels. It is a deep and a thoughtful approach to learning programming, mathematics, and problem solving techniques. Let me quickly say, however, that no background in mathematics is needed. The mathematics taught by Logo is at the same time both subtle and powerful

What *is* needed from you as a parent is that you follow your child's progress through the book and discover the main principles of Logo with him. In this way, you will be in a position to act as a mentor for your child, someone who can come to the rescue on occasion, or who can offer a special suggestion that will lead to exploration and discovery at a critical time.

At the end of the book, you will find Appendix A, which will describe how to use Logo with your child and outline the philosophies of Logo and Piaget, the Swiss psychologist whose ideas influenced the creators of Logo.

Most of all, Logo is intended to be fun. With Logo the child will have positive experiences in a mathematical and computer environment, an environment that is becoming increasingly important in our high-technology world.

Chapter 1



Your Very Own Turtle

Welcome to Logo and turtle geometry. Deep inside your computer, there is a tiny silicon chip that controls all of the things that your computer does. Sometimes, this computer chip is so good at its job that it can seem almost "smart." We are going to imagine that there is a turtle inside the computer instead of a chip. If the computer is going to seem smart, then the turtle is going to seem "smart" as well!

We are going to call the turtle "Sam," and we are going to tell Sam what he has to do. On your screen Sam will look like a little triangle, but before Sam looked like a little triangle, he was a real turtle, well . . .if not a real turtle then a robot turtle at least. We will still think of him as a real turtle though. As we move on, you will find that Sam really does behave exactly like a turtle. In fact you will find that he is an obedient little chap, as well as being a friendly fellow.

TELLING SAM WHAT TO DO

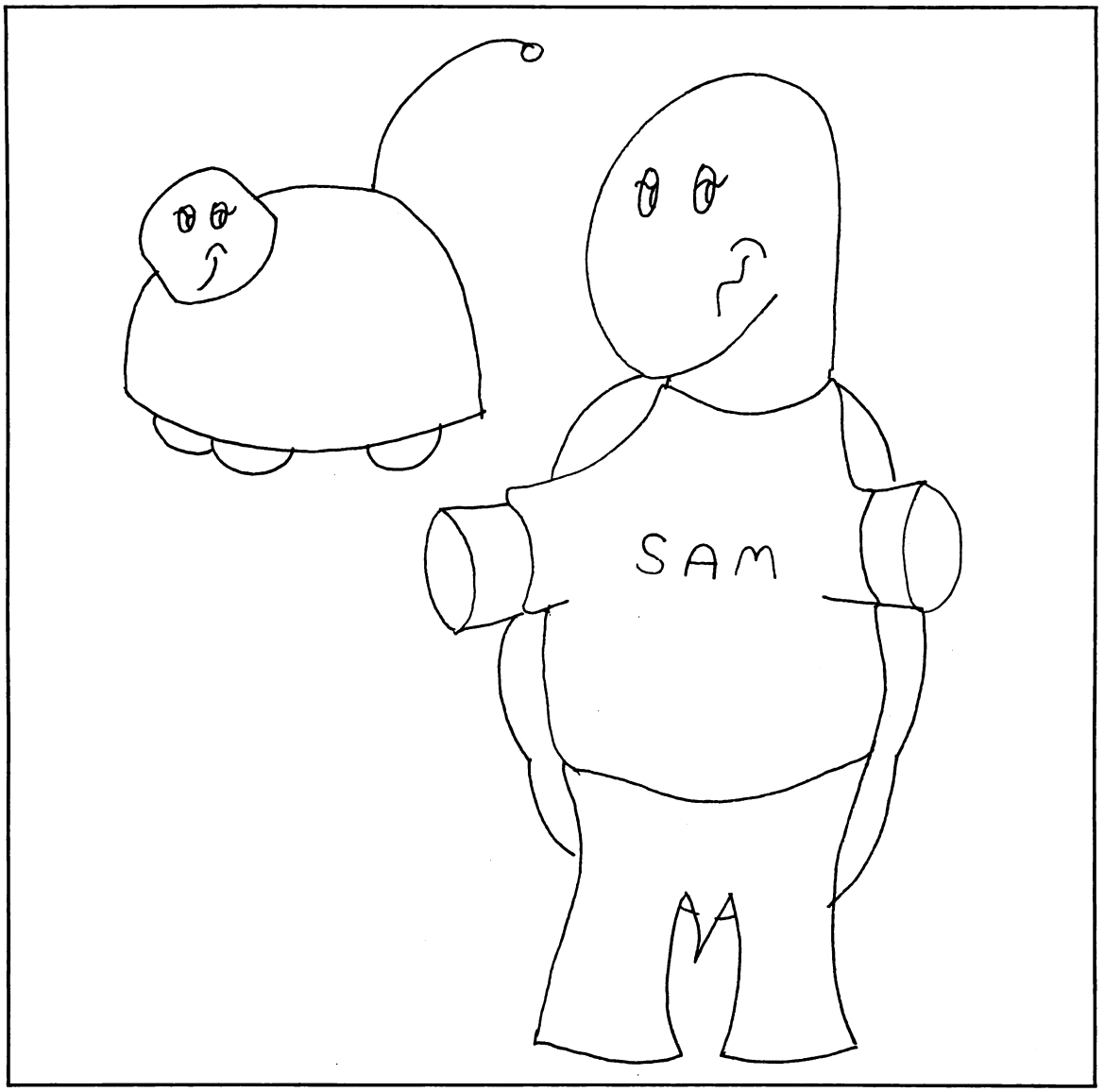
If you do not know how to start up the Logo language, then please refer to Appendix B or to your Logo manual.

At the bottom of the screen you should see a question mark. After the question mark you will see a flashing block. The question mark is called the *prompt* and the flashing block is called the *cursor*. The prompt and the cursor are the computer's way of telling you that it is ready for some instructions.

You can see that Sam is pointing upwards. He is facing north. The exciting thing about Sam is that we can tell him what to do. We can tell him to go forward, backward, right, left and many other things as well! Let's look at how to tell Sam to go forward!

TELLING SAM TO GO FORWARD

One of the simplest things that we can tell



Meet Sam our computer turtle.

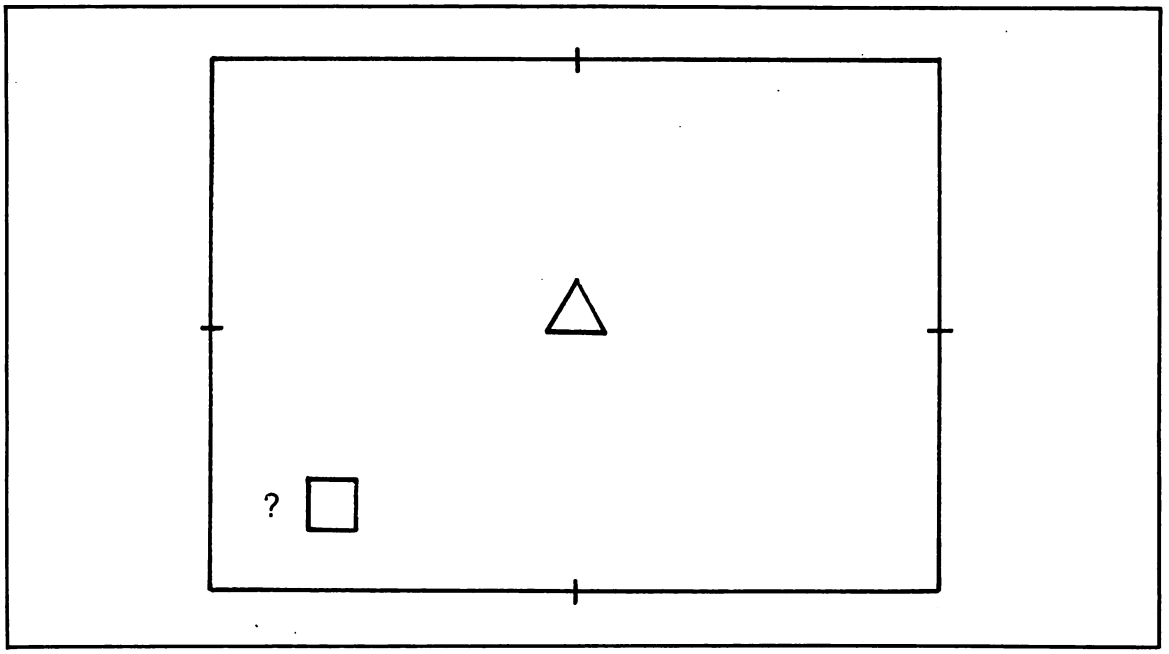
Sam to do is to go forward. Try typing in the commands that appear below. The word *return* in the brackets means that you press the [RETURN] key. The return key is on the right-hand side of the keyboard. Look at the next illustration to see where the return key is.

We can tell Sam to go forward by typing in the following

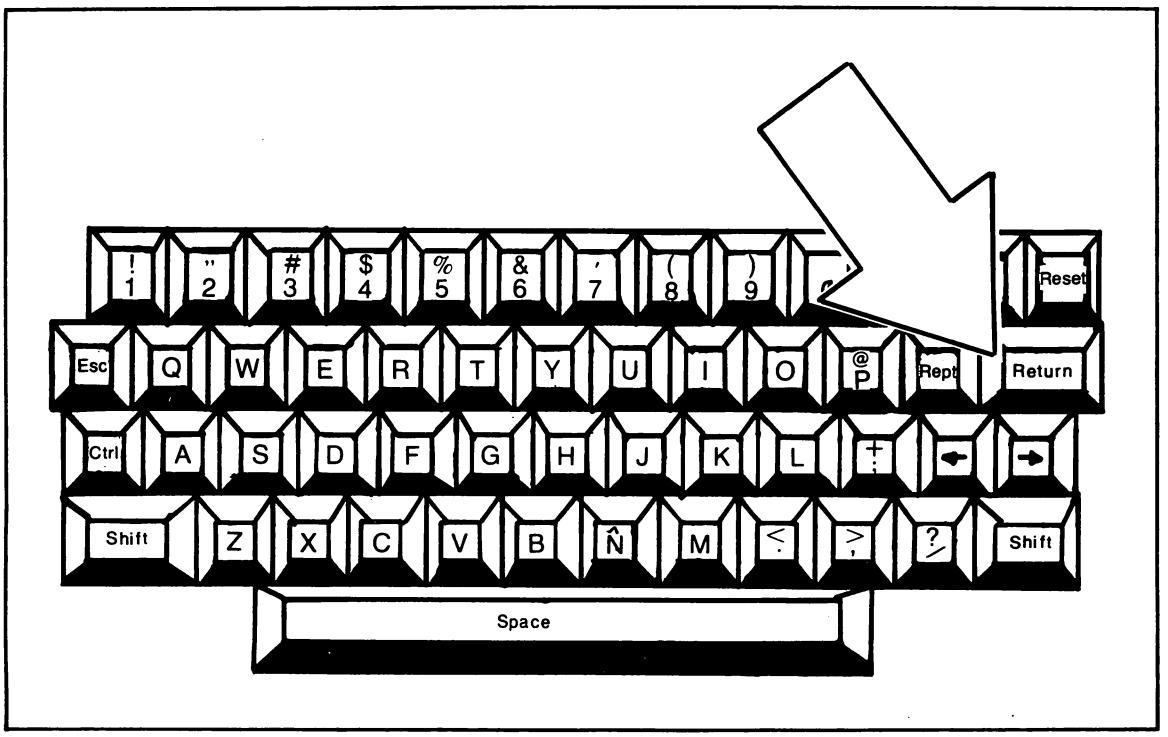
```
FD 50 [RETURN]
```

Sam walked forward 50 steps! He stopped part of the way up the screen. As you can see, FD stands for the word ForwarD. The number 50 told Sam that he should go forward 50 turtle sized steps.

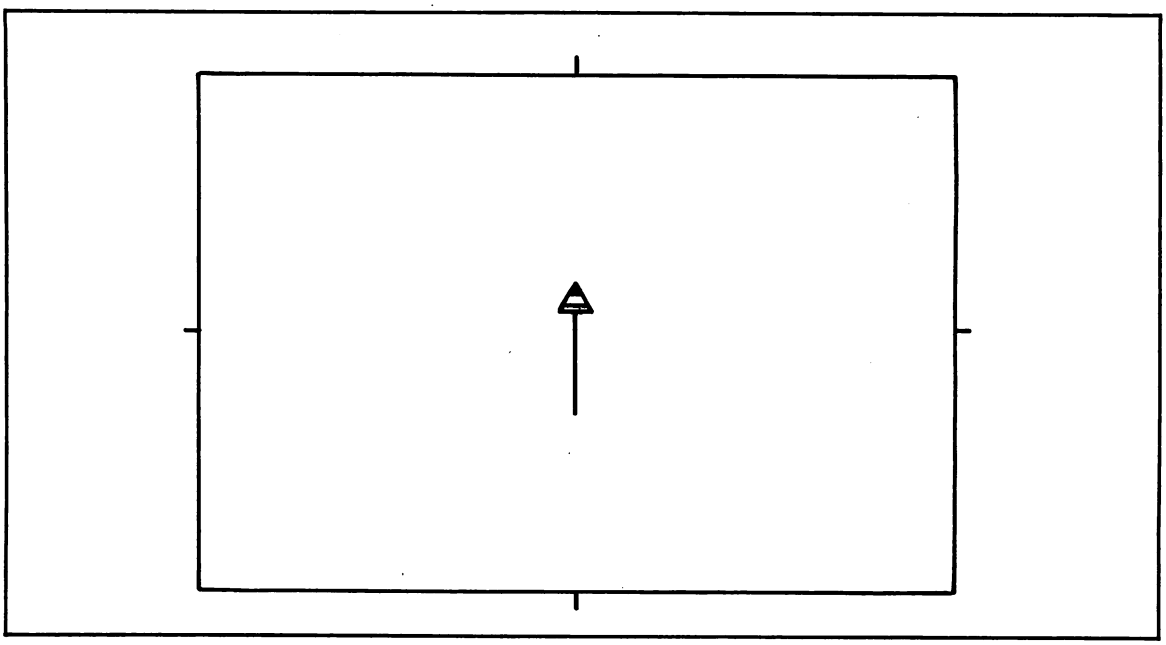
We should start over again with an empty



Prompt and cursor.



Here is the [RETURN] key.



FD 50.

screen! To wipe the screen clean, type the letters CS. Guess what CS stands for? It stands for ClearScreen! Whenever you want to wipe the screen clean, simply type the letters CS. When we type CS, Sam will go to his *home position* at the middle of the screen. When Sam is in the home position, he is at the center of the screen facing north.

Try typing this

```
CS
FD 100  [RETURN]
```

Sam walked forward again. This time he walked forward 100 turtle steps.

FD is a *primitive*. A primitive is a simple statement that tells Sam what to do.

POOR SAM DOESN'T UNDERSTAND

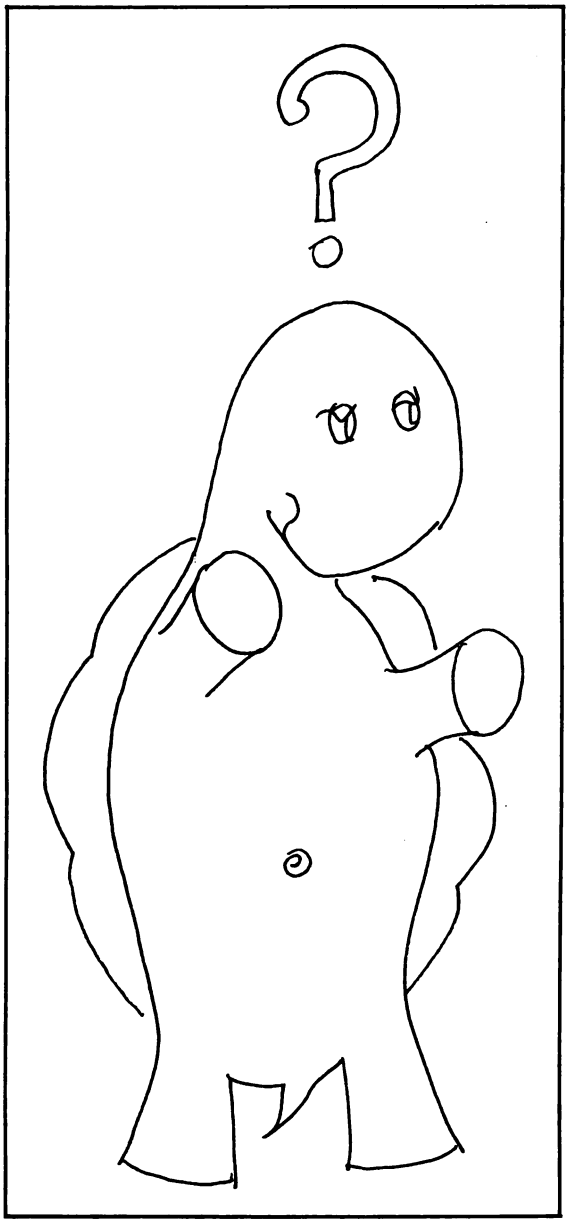
Have you noticed that you have to tell Sam exactly what to do? Although he is a smart little fellow, he really needs very exact instructions every time. After all, Sam really is a computer and computers aren't really all that smart! For instance if you leave out the space in FD 70, Sam will tell you "I DONT KNOW HOW TO FD70." This is his way of letting you know that he didn't understand.

Computers are like that. If you don't tell a computer exactly what to do, in exactly the right way, you will get strange things happening. When this happens, you say that a program or a procedure has a *bug*. A bug is a mistake in the instructions that you have given Sam.

If you expect Sam to do one thing, and you find that he does something else instead, then Sam has a different kind of bug. When Sam has this type of bug, you often have to change things in your procedure so that the bug will disappear.

HOW FAR CAN SAM GO?

Here are two primitives for you to type in. The first will clear Sam's screen, set him in the



Sam doesn't understand FD50!!

middle, and make him face north. The second will tell him to go forward 200 turtle steps. Last time, when we used FD 100, he went forward 100 turtle steps.

```
CS
FD 200 [RETURN]
```

Oops! Sam ran off the top of the screen and reappeared on the bottom. This is called *wrap-around*. This happens when we tell Sam to go forward sometimes. If he falls off the edge of the screen, he will reappear on the other side of the screen.

Here is a challenge for you. How far does Sam have to go from his starting point to the edge of the screen? You can find out by telling Sam to go forward until he just disappears off the edge of the screen! Why not try this, and then look at the picture of the screen at the bottom of page 6 to find out how big the screen really is!

The picture shows the screen measured in turtle steps.

If Sam is at his home position, he will have to go forward about 120 steps before he will fall off the edge of the screen. If he were facing to the east, he would have to go forward about 140 steps before he fell off the edge of the screen!

You should play with the primitives FD and CS until you are familiar with what they can do.

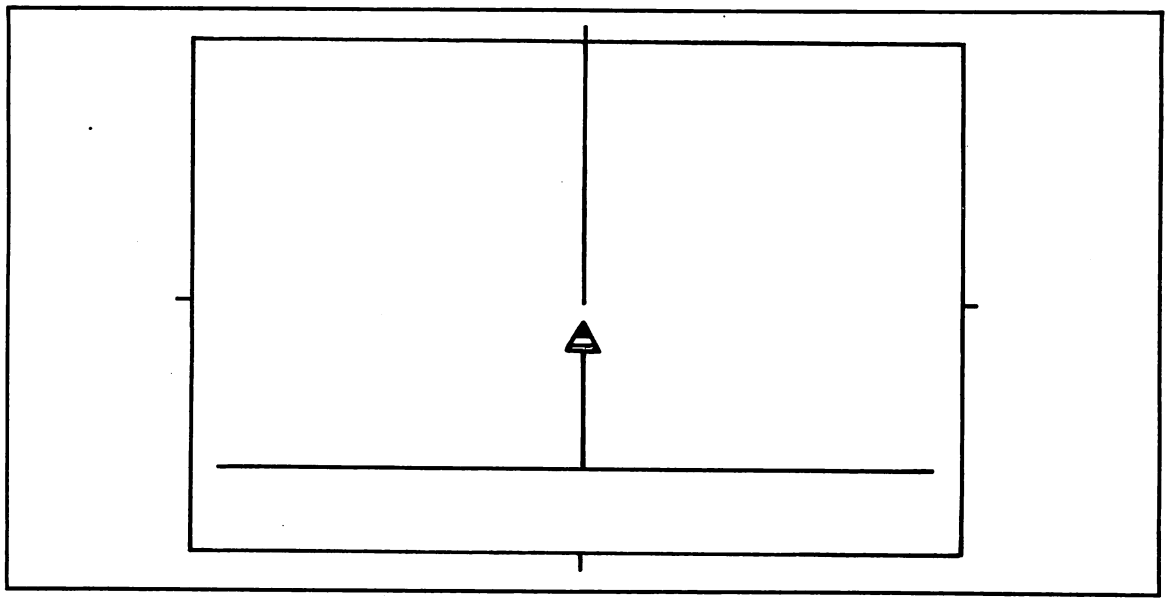
So much for telling Sam to go forward! I wonder if Sam can do anything else? Maybe we can tell him to turn right!

TELLING SAM TO TURN RIGHT

Let's tell Sam to turn a little! You can do this by simply typing:

```
CS
RT 90 [RETURN]
```

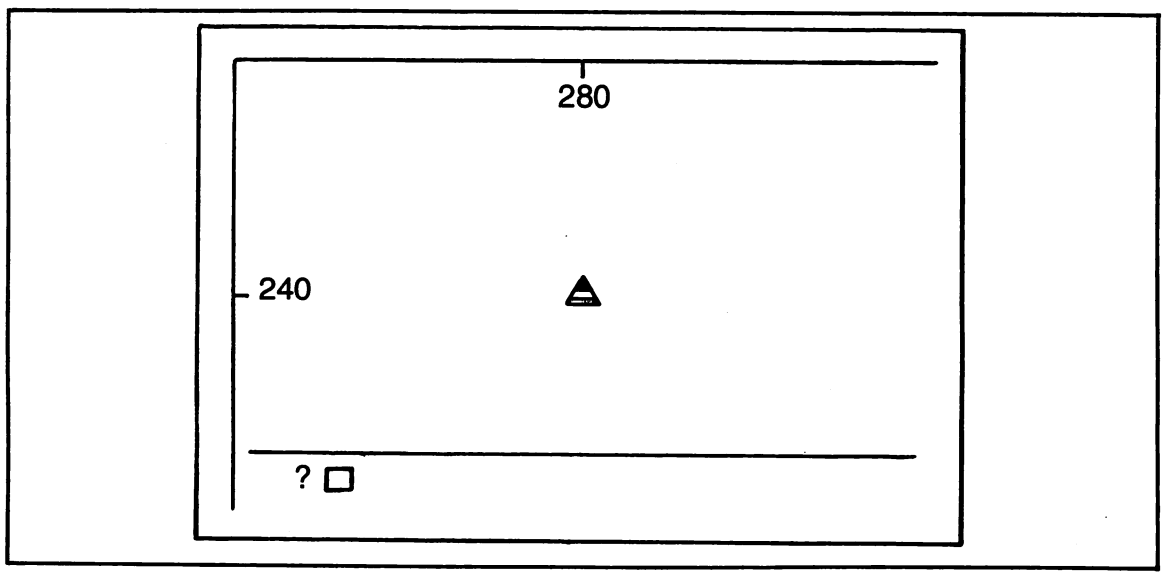
Sam turned from facing to the north to facing east. In fact, RT is short for the word Righ T. Page 8 shows the *points of the compass* that Sam uses when he is turning.



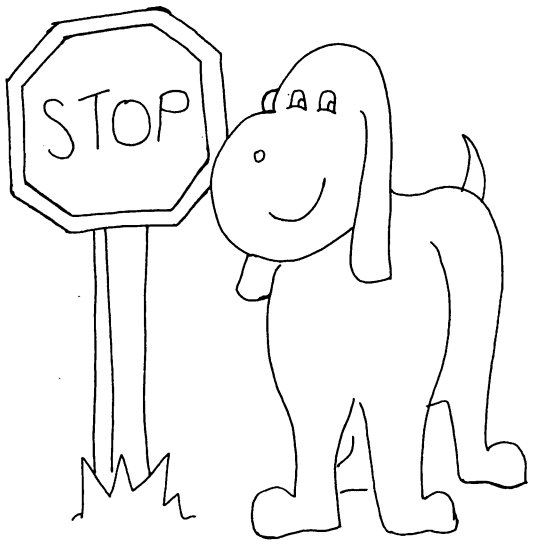
FD 200.

The number 90 will tell Sam how much he should turn. You still need the space between the RT and the number, so, if Sam tells you that "I DON'T KNOW HOW TO . . . . " check to see if you have missed any spaces.

Let's try to combine the two things that we know about so far, the FD and the RT primitives. By the way, from now on we will leave out the reminder to press the return key, as you are now aware that you should press the return key at the end of each Logo line or command.



Sam's screen.

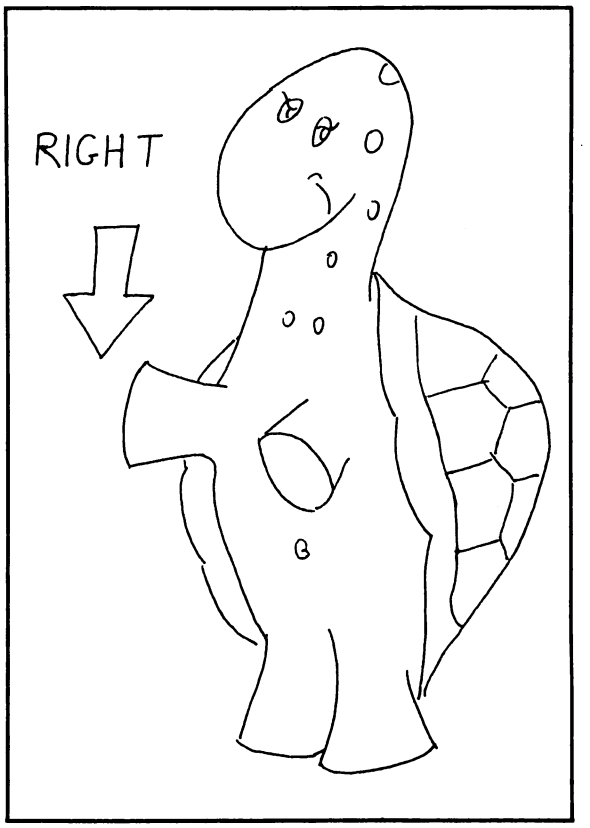


We will also leave out any CS primitives that are needed. Often you will want to use the CS command before you use a given procedure or primitive.

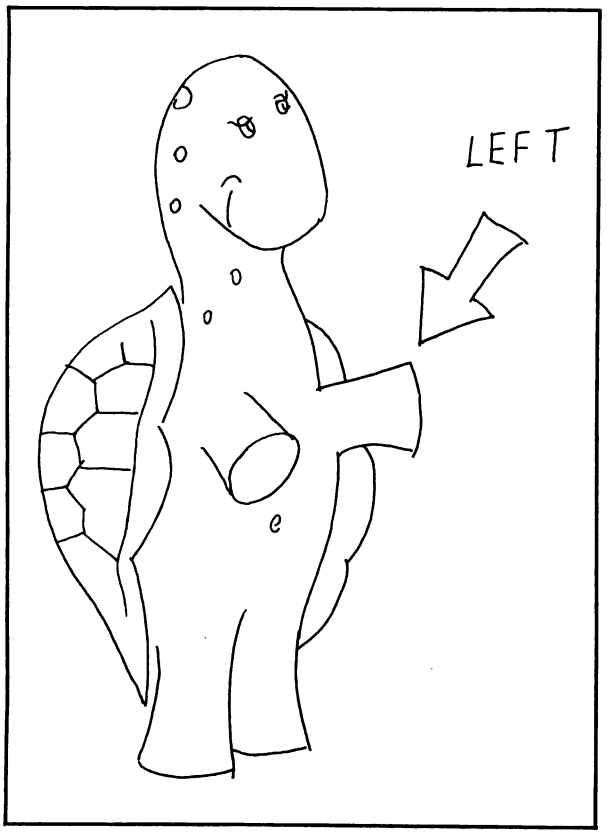
Lets put the two things that we know about together. Type in the following:

```
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
```

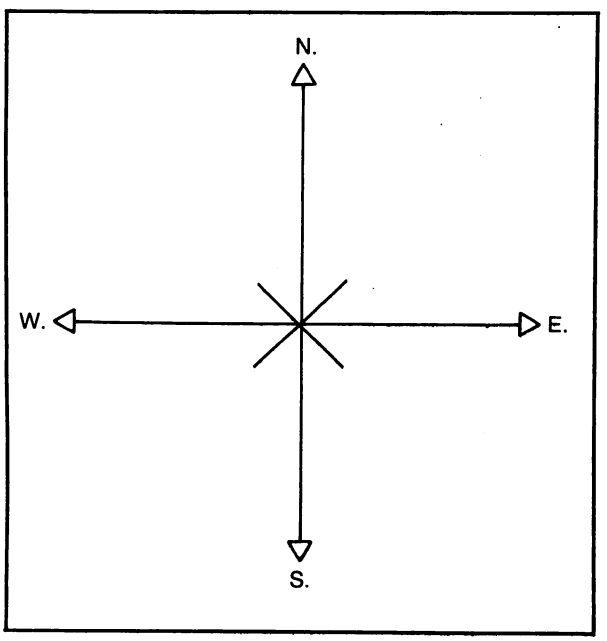
How about that! Sam drew a square. Or should I say that, you told him to draw a square.



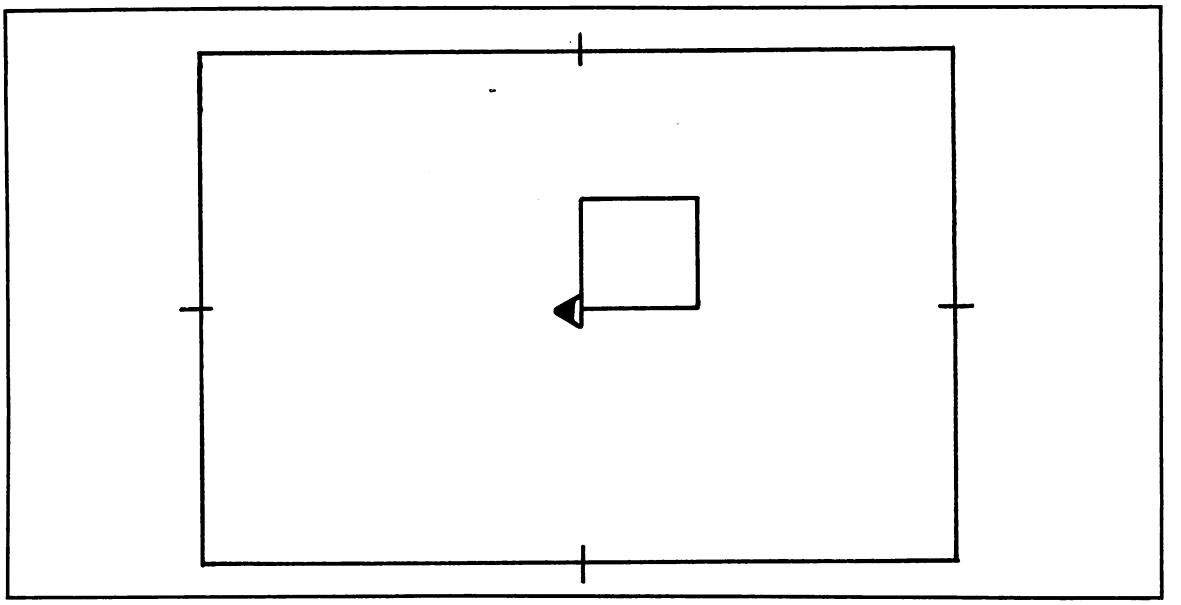
Sam and his right hand!



Sam and his left hand!



North South East and West.



Square.

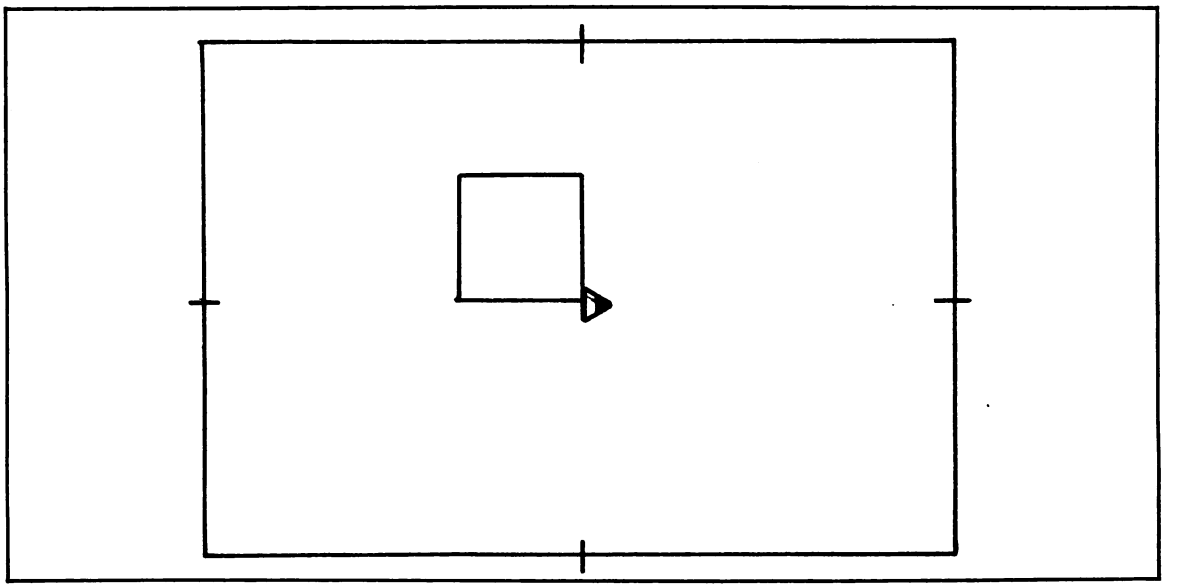
If we can tell Sam to go forward and right, I bet that he can turn to the left as well! A turtle isn't a real turtle if he can't turn left! Is he?

TELLING SAM TO TURN LEFT

While we are telling Sam to turn left, let's use some of the other procedures that we know about, at the same time, such as RIGHT and FORWARD.

Type in the following.

```
FD 50
```



Left Square.

```
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
```

How about that! We told him to square left this time.

I guess I don't have to tell you that LT 90 will make Sam turn to the left. If Sam is home and you tell him to LT 90 then you will see him turn and face to the west. If you used 45 instead of 90, then you would find that Sam turns half way between north and west. After all, 45 is halfway between 0 and 90! Of course, if we had used LT 45 instead of LT 90 this would not have given us a square, it would have given us a completely different pattern. Try telling Sam to go LT 45 instead of LT 90 to see a new pattern.

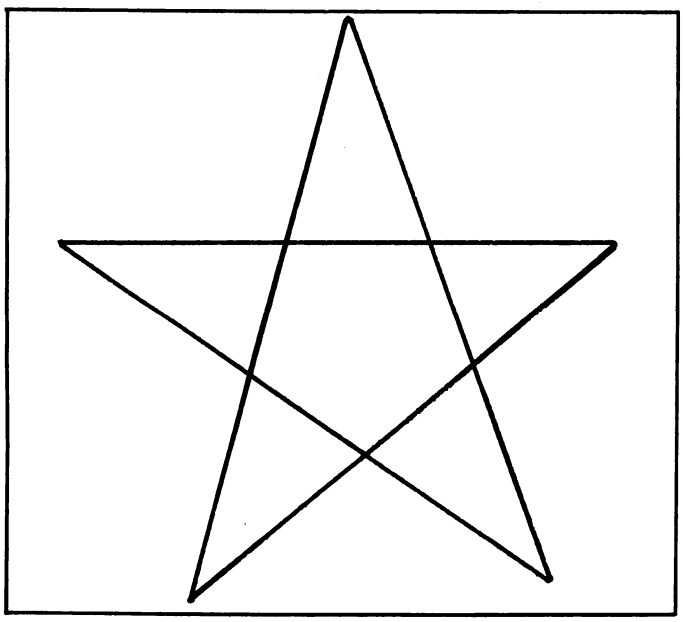
We can combine FD, RT, and LT in many ways. At the same time, we can use any numbers that we please with the FD, RT, and LT primitives. Every time you make a change to a primitive or a number, you will get a different pattern.

There are two really nice things about Sam. First, you can tell him what to do and second, you can change the numbers and primitives around to see what will happen to Sam's patterns.

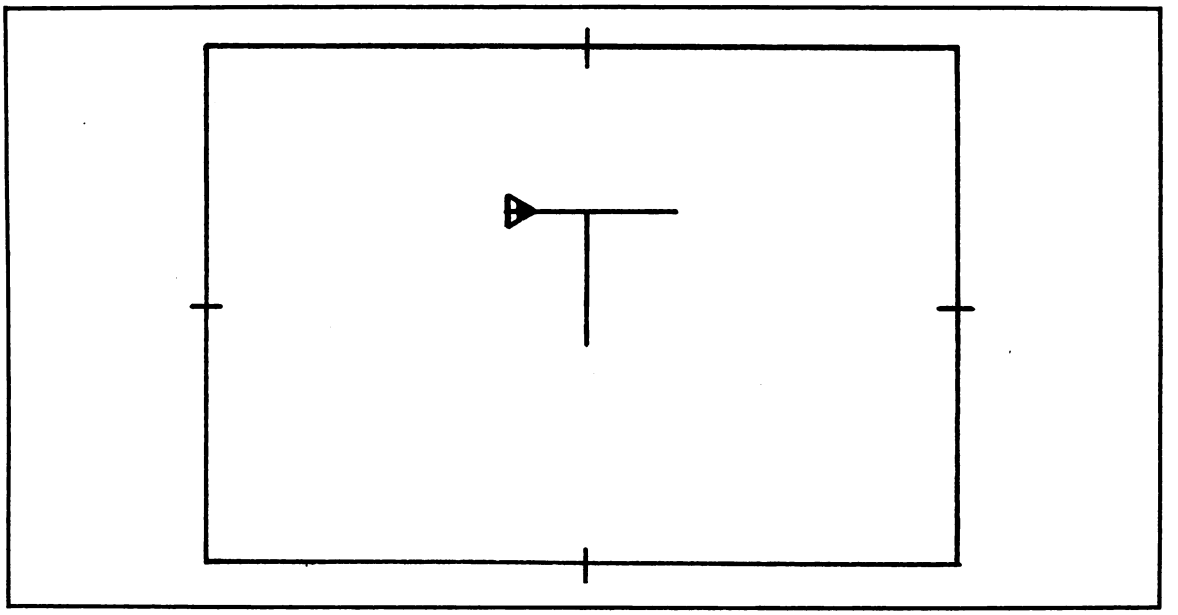
I wonder if we can tell Sam to draw a triangle? This can be quite difficult, because, we really would like to be able to close the triangle exactly. We will talk about how to do this in chapter 2. In the meantime, here is what you would have to tell Sam to have him draw the triangle.

```
FD 100
RT 120
FD 100
RT 120
FD 100
```

Were you surprised at the RT 120's? If you look at the screen, then you will see that



Star.

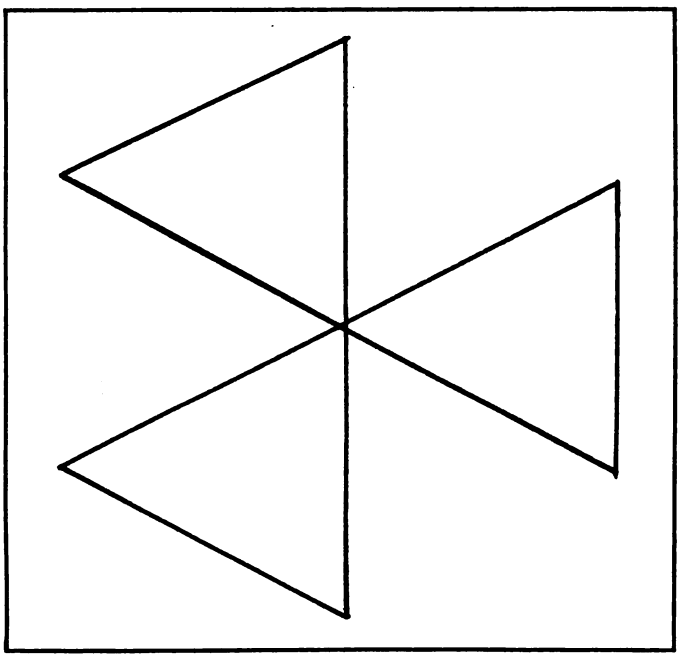


"T".

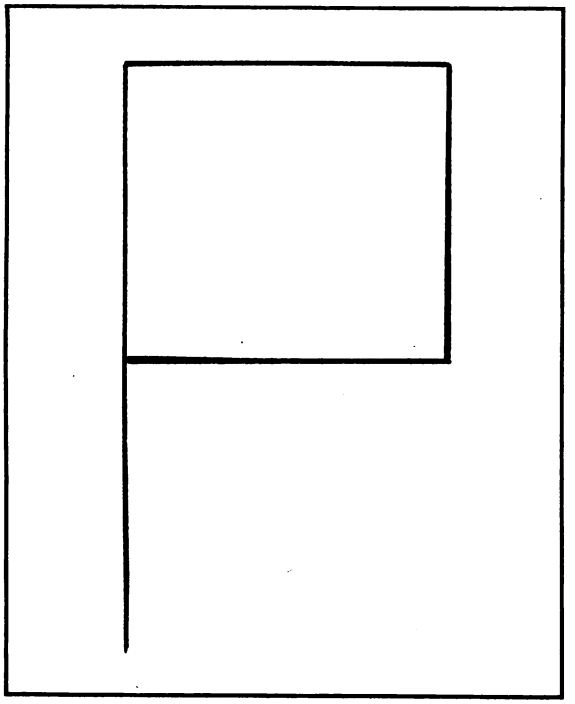
we will really need large number values like 120 if we want to draw something like a triangle!

Here is a list of primitives for you to type in that will give you a star on the screen.

```
FD 100
RT 144
```



Pattern.



Flag.

```
FD 100
RT 144
FD 100
RT 144
FD 100
RT 144
FD 100
```

SAM CAN BACK UP!

It seems that we can tell Sam to go forward and to turn right and to turn left, but how about telling him to go back? A smart turtle should be able to back up, shouldn't he?

Try typing in the following:

```
CS
FD 50
RT 90
FD 20
BK 40
```

Did you get the letter T on the screen?

The primitive BK is used just like the other primitives in this chapter. BK stands for BacK. If you tell Sam to BK 50, he will move back 50 steps. If you tell him to BK 80, then he will move back 80 steps. As he moves back, he will draw a line, of course, just as he did before.

Here is a list of primitives, which includes BK, RT, FD and LT, for you to type in.

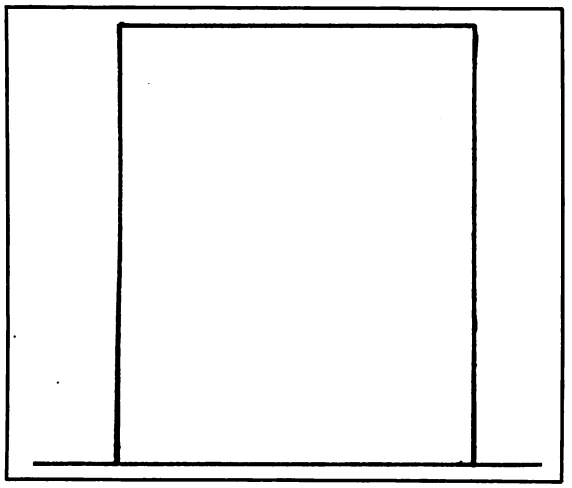
```
CS
FD 50
RT 60
BK 100
RT 60
FD 50
RT 60
BK 100
RT 60
FD 50
RT 60
BK 100
```

Page 11 shows an illustration of the pattern that you get.

Let's tell Sam to draw a picture of a flag. Type in the following.

```
FD 100
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
```

That was quite easy! We told Sam to go forward 150 steps, and then we drew a square to make the flag on the flag pole. You can draw many things in this way in Logo. Can you think of any that you would like to try?



Top hat.

SUMMARY

Look closely at the following summary, and then try the Quiz. The Quiz is almost the same as this Summary. If you find a question in the Quiz difficult, then simply look back at this summary; you will find all of the answers here!

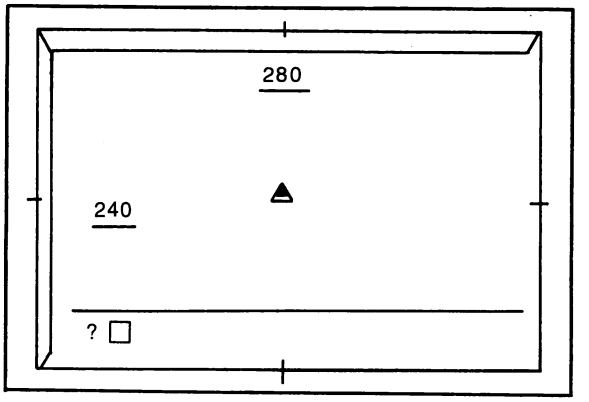
Just after the summary, you will find some challenges for you to try. It would be a good idea if you tried at least some of the challenges there.

At the end of the chapter, you will find a crossword puzzle, which has some of the new words and ideas from Chapter 1 in it.

1. Sam always does what he is told.
2. You must be careful to tell Sam exactly what to do.
3. It is important to include spaces and punctuation in your primitives so that Sam can understand what you mean.
4. You can tell Sam to do the following:

```
FD 30  (go ForwarD 30)
RT 90  (turn RighT 90)
LT 20  (turn LefT 20)
BK 40  (go BacK 40)
CS     (ClearScreen)
```

5. You must always press the return key after typing a Logo primitive or command.
6. The dimensions of Sam's screen are:

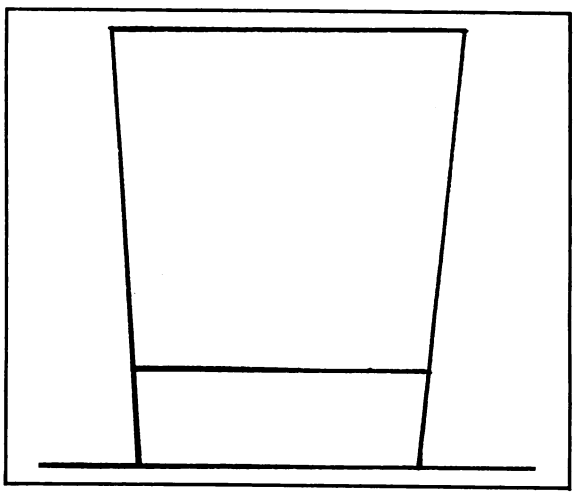


Screen size.

7. When Sam walks off the edge of the screen and reappears on the other side, Sam is said to *WRAPAROUND*.

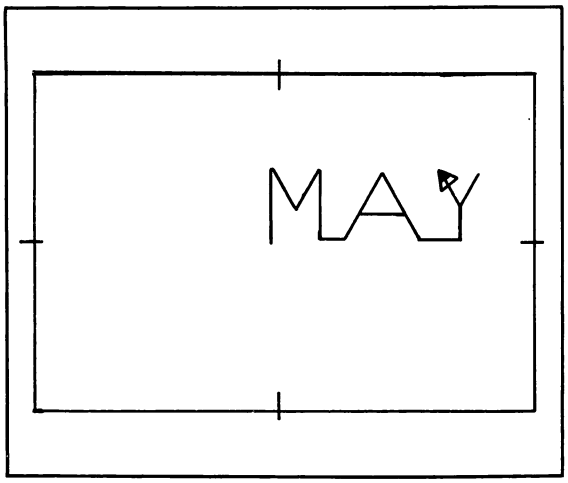
CHALLENGES

1. Tell Sam to move around on the screen using FD, RT etc. Don't worry about drawing anything this time, simply practice to become familiar with the new primitives that you know.



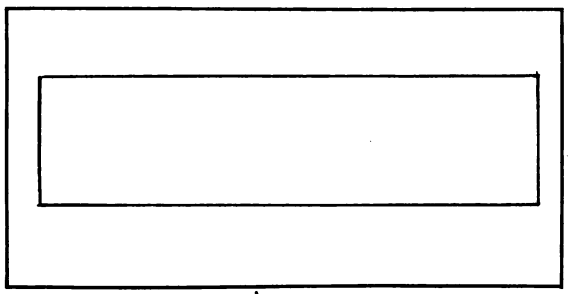
Difficult but better top hat.

2. Tell Sam to go FD, RT, BK etc. so that he will draw *your* initials on the screen.



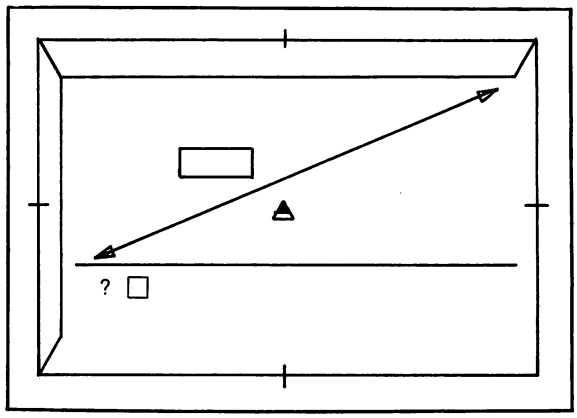
MAY.

3. Tell Sam to draw the rectangle shown.



Rectangle,

4. Choose one of the top hats on page 13 and tell Sam to draw it. If you choose the second one and it is too difficult, try the first one instead.
5. Using the primitives from Chapter 1, tell Sam to draw a box airplane at the center of the screen. You can use the illustration on page 15 as a pattern if you like.
6. Tell Sam to draw a flat-roofed building with doors and windows as shown on page 15.
7. How many diagonal steps are there from one corner of the screen to the other?



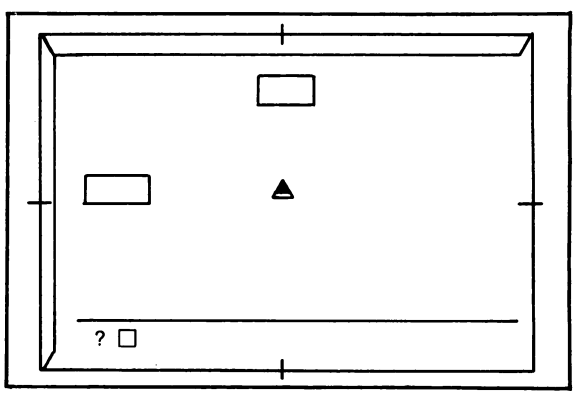
Screen Size.

QUIZ

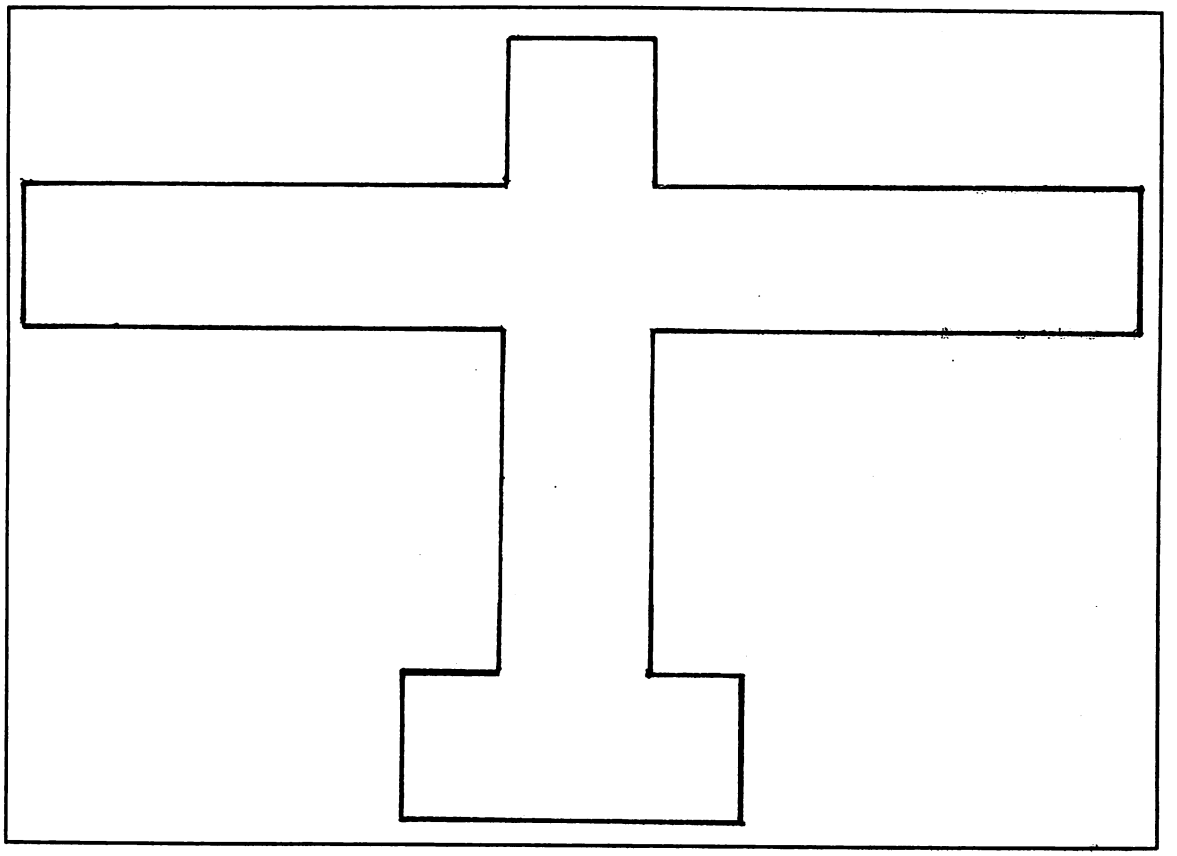
1. Sam always does what he is _____.
2. You must be careful to tell Sam _____ what to do.
3. It is important to include _____ and _____ in your primitives so that Sam can understand what you mean.
4. You can tell Sam to do the following:

FD 30	_____
RT 90	_____
LT 20	(turn LefT 20)
BK 40	_____
CS	(ClearScreen)

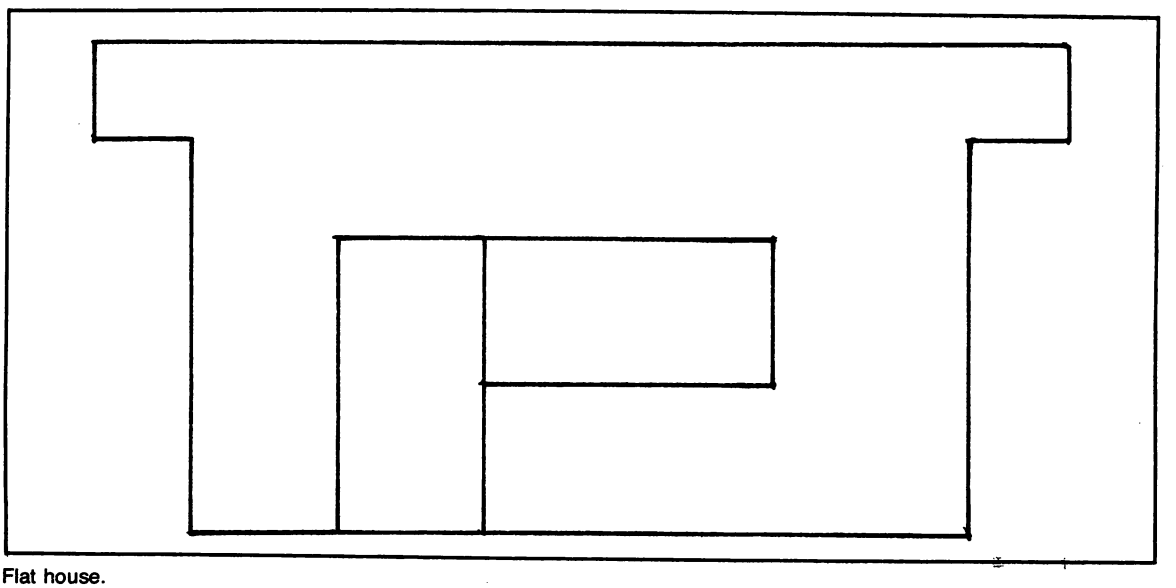
5. We always press the _____ key after typing a Logo primitive or command.
6. The dimensions of Sam's screen are:



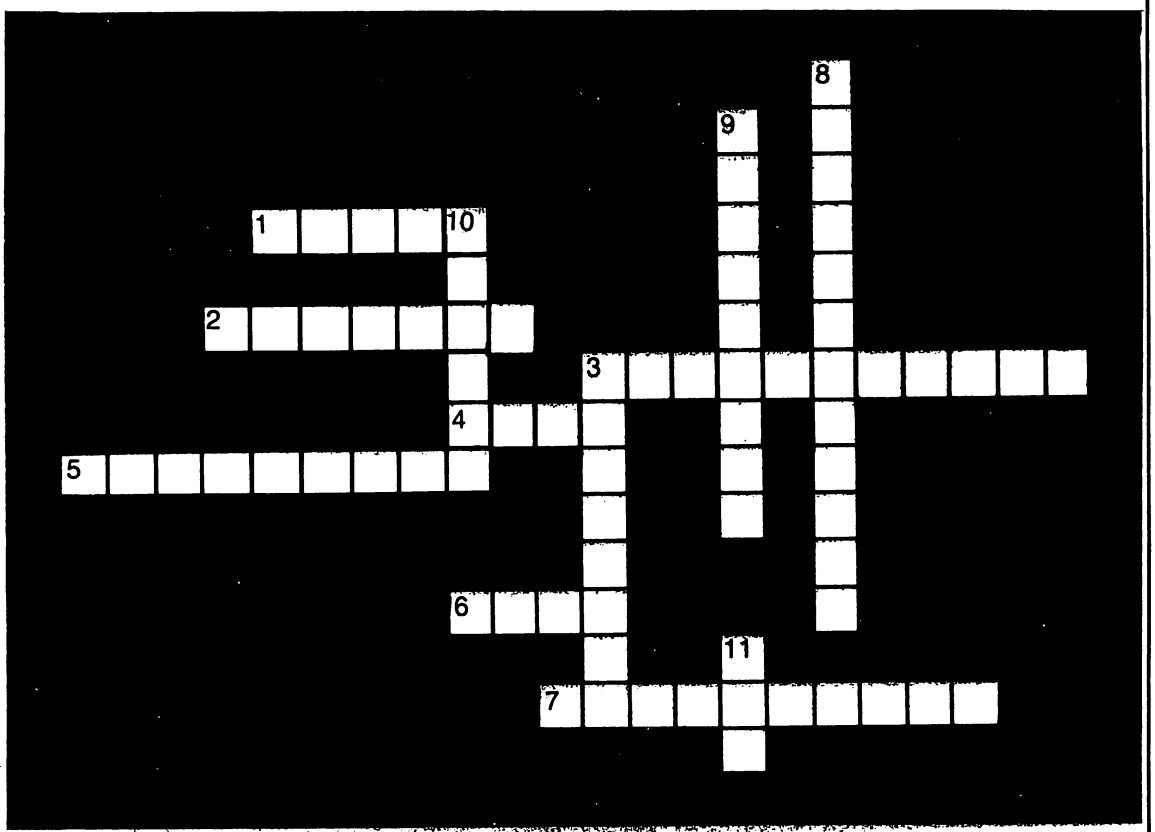
Screen size.



Plane.



Flat house.



7. When Sam walks off the _____ and reappears on the other side, Sam is said to _____.

CROSSWORD PUZZLE

Clues Across

1. In Logo, RT will tell the turtle to go _____.
2. In Logo, FD will tell the turtle to go _____.
3. CS stands for _____.
4. _____ This is a language that Sam understands!
5. A short Logo word that tells Sam to do something is called a _____.
6. LT will tell Sam to turn _____.
7. When Sam goes off the right side of the screen and reappears on the left side, this is called _____.

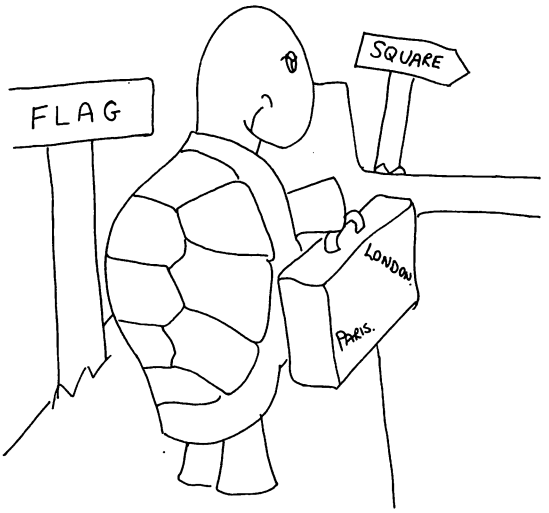
Clues Down

8. Sam is here when you start up Logo.

9. BK is a Logo primitive that will tell Sam to go _____.
10. Even though Sam is just a triangle on the screen, at one time he was a real _____.

3. Sam lives deep in the heart of our _____.
11. _____ This is someone who loves to be told to do things.

Chapter 2



Teaching Sam How To Do Things

As well as being very obedient, Sam is really quite smart in his own turtle way. You can *teach* Sam the meaning of words. For example you can teach Sam the meaning of the word square. Of course, Sam doesn't really know what a square is, but we can make it seem as though he does.

Let's see if we can teach Sam the meaning of the word square!

TEACHING SAM HOW TO SQUARE

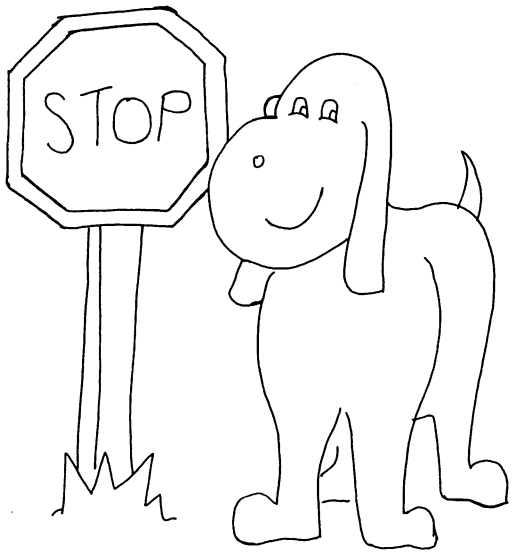
To make Sam understand the meaning of the word square, type the following.

```
TO SQUARE
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
END
```

There are two new items here. The first is the command TO SQUARE and the second is the word END. When you use the word TO, you are creating a *PROCEDURE*. A procedure is a list of things for Sam to do. This means that a procedure is really a list of primitives. The TO statement is always followed by the name of your procedure. You get to choose the procedure name! You can choose any name that you want, but you cannot use words like FORWARD and RIGHT that already mean something to Sam!

We could have named the procedure that draws the square almost any name. For instance we could have called it FLOWER. The turtle would still draw the same thing of, course; that is, a square! As we said earlier,

Sam doesn't really know what a square is, not in the same sense that you and I know what a square is.



When you choose the name for your procedure, be sure that you choose something that will help describe what the procedure does. For instance, if our procedure was to draw a house, then we would call the procedure HOUSE. Or if our procedure was to draw a block, then we would call the procedure BLOCK and so on.

Sam doesn't have the type of understanding that we have. Sam simply goes through the steps that we type into the computer, regardless of the name we give those steps.

It is something like watching a bug in the sand. If by chance the bug happens to trace the outline of a square, we wouldn't assume that the bug knows anything about squares. We can tell Sam what to do, and how to do it, but just because he can remember these steps doesn't mean that he understands what the steps mean.

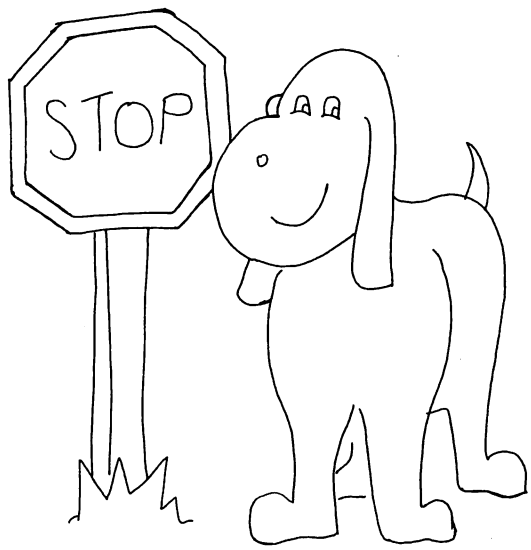
In Logo, END is a special word, it marks the end of a procedure. There is always an END statement to finish the definition of a procedure. Sam is a very helpful turtle. He will sometimes add the word END to your procedure, if you happen to forget it!

When you type **END** and press the [RETURN] key, Sam will tell you that **SQUARE IS DEFINED.** This means that Sam now knows the meaning of the word SQUARE.

If you want to see Sam draw the square, then type this at the keyboard.

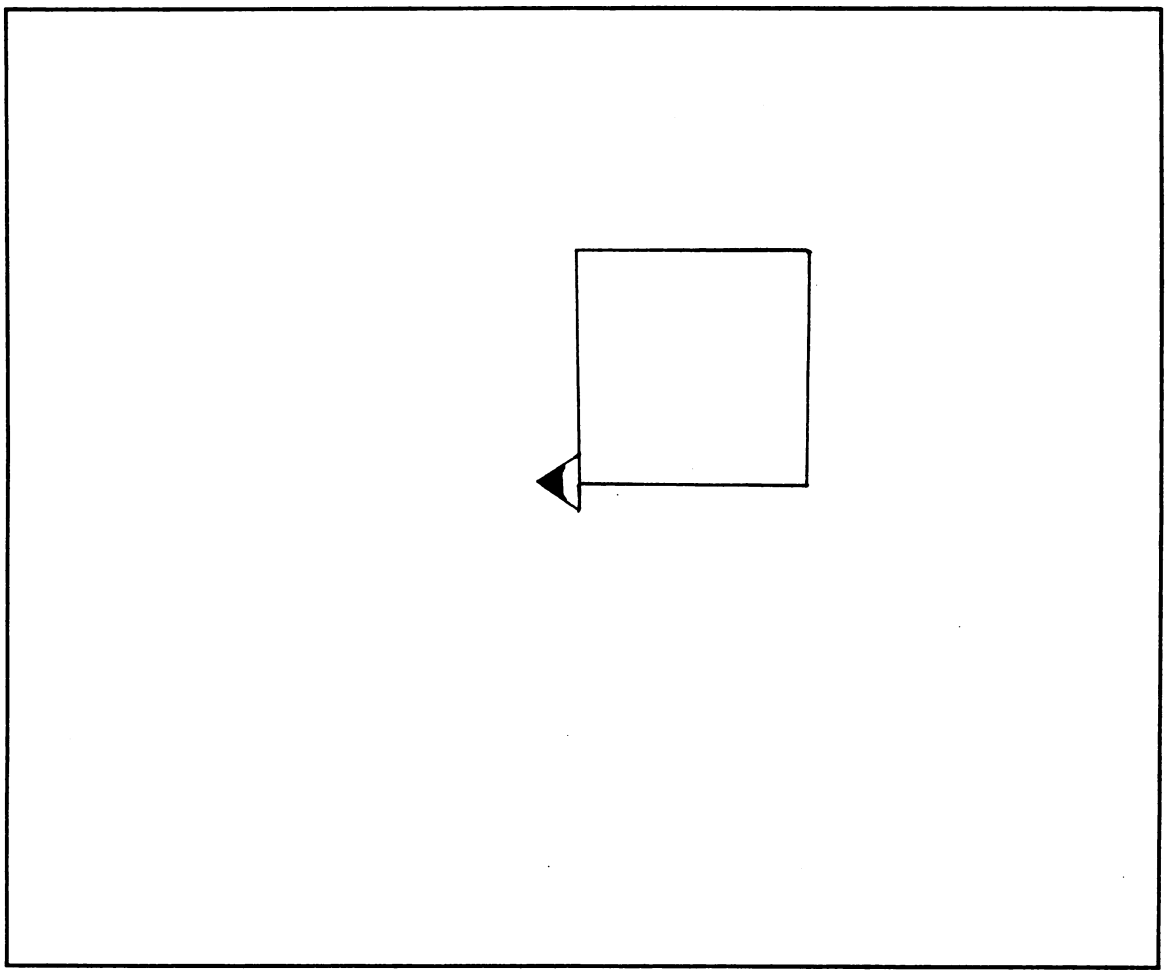
```
SQUARE
```

When you press [RETURN], you should see the square shown on the next page.



Do not switch off your computer at this point in the chapter. The next few procedures depend on Sam's knowing the meaning of SQUARE, and FLAG. If you switch the machine off, Sam will forget these procedures and you will have to type the procedures in again at the keyboard.

We will talk about how to avoid this a little later on in this chapter.



Square.

Here is another procedure for you to try. You will find that this procedure will draw a flag. Of course a flag is like a square. It is a square with an extra line drawn for the flagpole. The FLAG procedure will have the SQUARE procedure inside it.

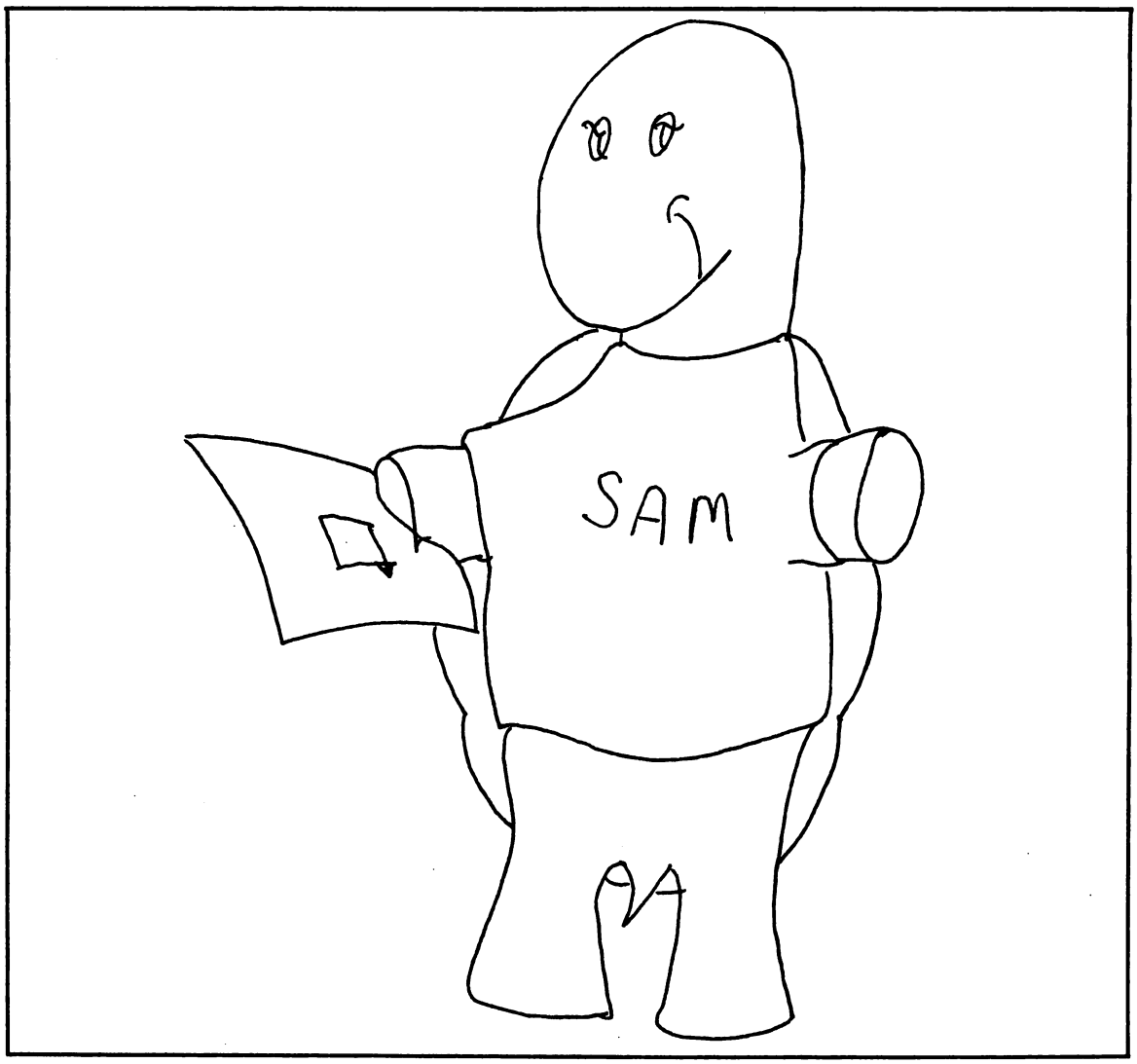
```
TO FLAG
FD 50
SQUARE
RT 90
BK 50
END
```

To see Sam's flag, type the following at the keyboard.

```
FLAG
```

Page 23 shows an illustration of what you should see on the screen when you type the command FLAG.

The SQUARE procedure was called from within the FLAG procedure. You will find that this idea of a procedure calling a procedure is used all of the time in Logo. What is happening is this. Sam begins a procedure by looking at



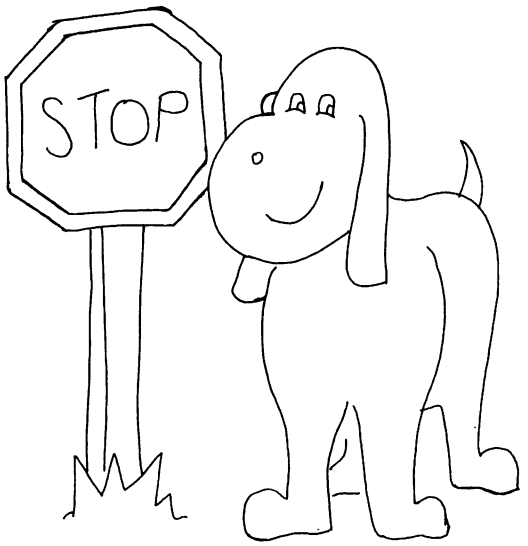
Sam knows how to square!

the first primitive and obeying the instruction that it gives him. After that he goes to the next primitive, and then obeys that instruction. If he comes to an instruction that is the name of another procedure, then he takes a side trip and goes to that procedure so that he can obey the primitives there. When he is finished, he will return to the first procedure and continue from where he left off.

Here is another example of a procedure that uses FLAG as a subprocedure.

```
TO THREE.FLAG
FLAG
RT 30
FLAG
RT 30
FLAG
END
```

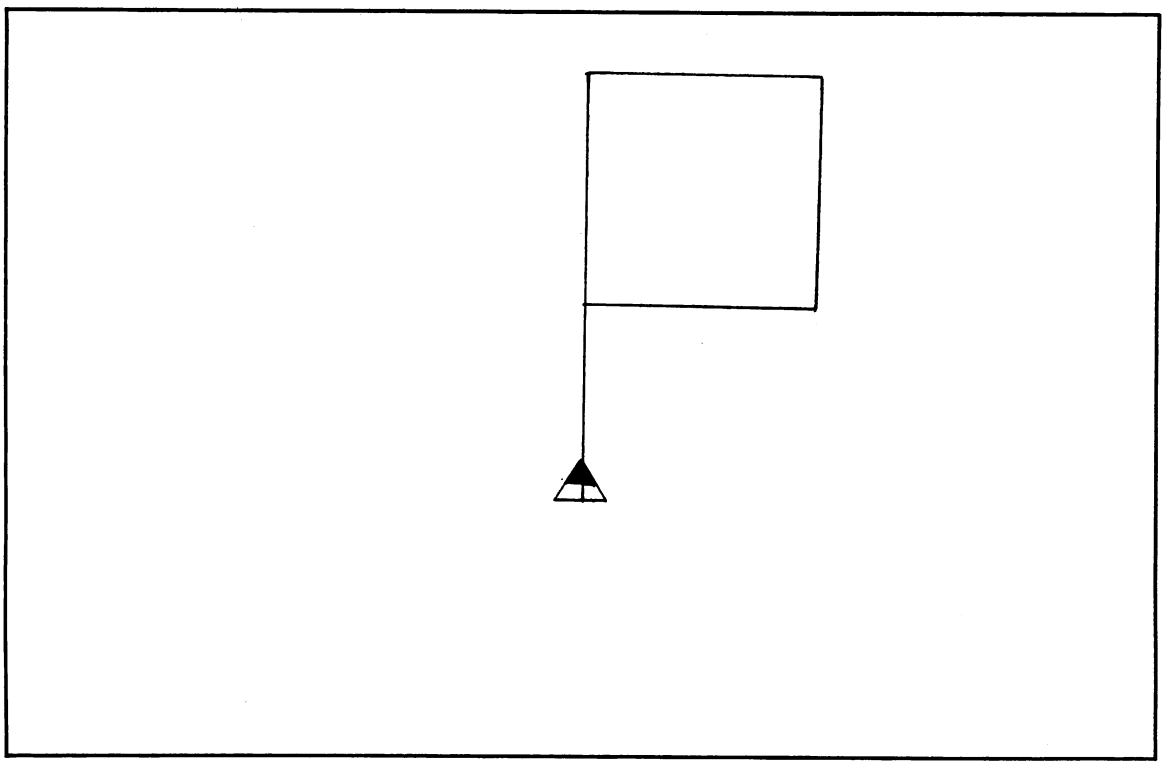
Type THREE.FLAG to see Sam draw the



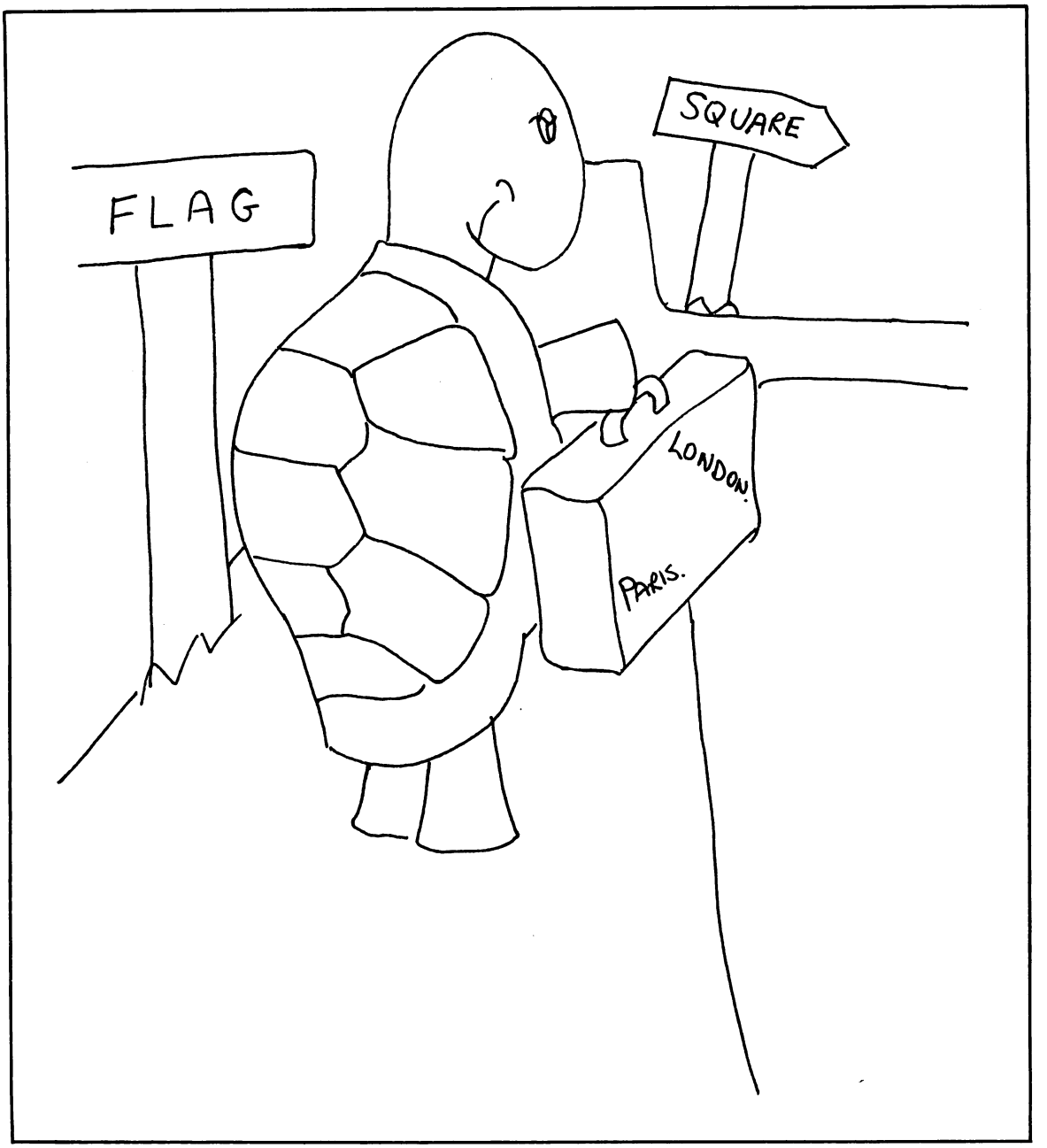
flags. Your screen should look like the one on page 25.

Did you get the message I DON'T KNOW HOW TO FLAG or I DON'T KNOW HOW TO SQUARE? If you did, check that Sam still has the procedure FLAG in his memory! You can do this by typing the word FLAG and the word SQUARE one at a time into the computer. If Sam tells you that I DON'T KNOW HOW TO FLAG or I DON'T KNOW HOW TO SQUARE, then for some reason, he has lost the procedure FLAG or SQUARE. You will have to retype the SQUARE and FLAG procedures if this has happened.

The procedures FLAG and THREE. FLAG use the procedure SQUARE to draw the shapes they need. Sam needs to know the procedure SQUARE before he can call it for use in the FLAG procedure.



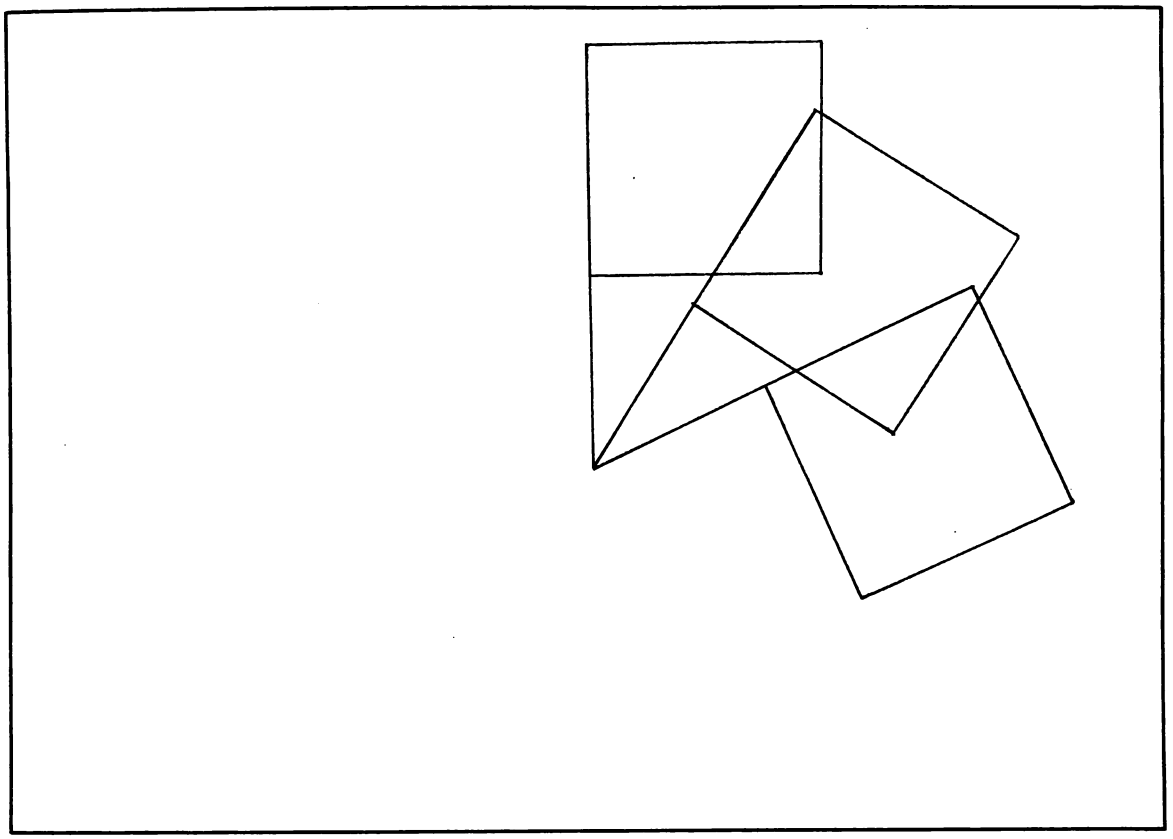
FLAG.



Sam takes a side trip.

If you have just started your computer, and you want to see the THREE.FLAG procedure run, you would have to type in all of the following.

```
TO THREE.FLAG
FLAG
RT 30
FLAG
```



THREE.FLAG.

```
RT 30
FLAG
END

TO FLAG
FD 50
SQUARE
RT 90
BK 50
END

TO SQUARE
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
END
```

When you have finished typing the procedure, then you would have to type THREE.FLAG to see the procedure executed.

This time, the procedure FLAG was called from the procedure THREE.FLAG. At the same time, the procedure FLAG also had to call the procedure SQUARE. The procedure SQUARE was used of course in making the flag.

We now know how to TELL Sam and how to TEACH Sam, but we have to type in Sam's procedures each time we switch the machine on. What about the disk drives? Can we save

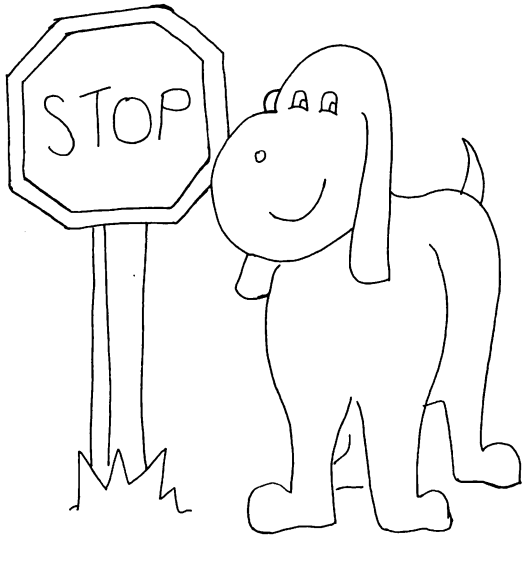
our procedures on them somehow? If we could, then we might be able to save ourselves some typing!

SAVING YOUR PROCEDURES ON DISK

Most of us don't type well, and being able to program a computer and telling Sam what to do hasn't very much to do with typing. It would be nice to avoid most of the typing chores. Have you noticed that most of the primitives in Logo are only a few letters long so that you will not have much to type when you are telling Sam to do things?

When we switch off the computer, Sam loses his memory. Even when we restart Logo, Sam will not remember what we had taught him before we switched off the machine. We would have to retype Sam's procedures at the keyboard if we wanted him to understand the meaning of our procedures once more!

Thank goodness we have a command called SAVE! The SAVE command will allow us to save our procedures on the disk. We should make a point of saving our procedures before we end a session on the computer. When we start a new session, we will load the procedures into Sam's memory from the disk. This process will save us a lot of typing. So far, we haven't used the disk drives. Let's use the drive to save and reload our new THREE. FLAG procedure to our file disk.



If you have not already prepared a file disk, go to Appendix B to find out how to do this. If you don't want to use a second disk for your procedure, you can simply save your procedures on your Logo program disk. It is however, much better if you use a separate file disk!

To save the file on disk, type the following.

```
SAVE "THREE.FLAG
```

The drive will whir and chatter while your procedure is written onto the disk. A new filename will now appear in the disk's catalog. The catalog is a list of all the items that are on the disk. To check that your THREE.FLAG procedure is on the disk, type the following word.

```
CATALOG
```

You should see the file THREE.FLAG.LOGO listed on the screen. Logo added the word .LOGO to the filename so that you will know that THREE.FLAG is a Logo file.

Now that our THREE.FLAG procedure is safely stored on our disk, let's look at something else. I wonder what we can look at? I know, we know how to put things into Sam's memory from the keyboard, but how about erasing things from Sam's memory?

HOW TO ERASE SAM'S MEMORY

We already know that once Sam has been taught the meaning of a procedure, he will hold

```
DISK VOLUME 254

 A    6 HELLO
 T    6 STARTUP.LOGO
 T    2 SQUARE.LOGO
 T    2 THREE.FLAG.LOGO
 T    2 TRIANGLE.LOGO
 T    2 FLAG.LOGO
 T    2 TRI.LOGO
 T    2 PATTERN.LOGO
 T    2 FOUR.SQUARE.LOGO
 T    2 EIGHT.SQUARE.LOGO
 T    2 SQUARE1.LOGO
 T    2 WINDMILL.LOGO
 T    2 SKINNY.RECTANGLE1.LOGO
 T    2 SKINNY.RECTANGLE.LOGO
 T    2 MANYLEG.LOGO
 T    2 COLOR.LOGO
 T    2 BLUESQUARE.LOGO
 T    2 BLUESQUARE2.LOGO
 T    2 RANDOM.COLORS.LOGO
 T    2 DOUBLE.SQUARE.LOGO
 T    3 SCREEN.WRAP.DEMO.LOGO
 T    3 SCREEN.MODE.DEMO.LOGO
 T    2 CIRCLE.LOGO
 T    2 FOUR.CIRCLE.LOGO
 T    2 FLOWER.LOGO
 T    2 SPIN.CIRCLE.LOGO
 T    2 BETTER.FLOWER.LOGO
 T    2 WIGGLE.WAGGLE.LOGO

 ?
```

Catalog.

that procedure in his memory until the machine is switched off. Sometimes, this can be very inconvenient. You might change your mind about keeping the procedures that you have created. You might even want to start all over again with Sam's memory completely blank.

You still have the THREE.FLAG procedure in Sam's memory. If you try to call another procedure by the name THREE. FLAG, you will find that Logo will not let you use the same name twice. This is another reason why you might want to erase a procedure.

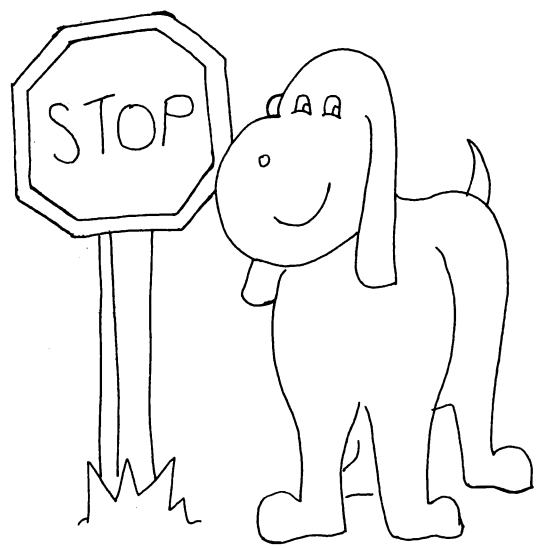
Let's make Sam's memory a complete blank! Type in the command.

ERPS

ERPS stands for ERaseProcedureS. If you type the command ERPS you will rub out *all* of Sam's memory.

Just to check that everything is gone, try typing THREE.FLAG. Sam will tell you I DON'T KNOW HOW TO THREE.FLAG.

That of course is correct; after all we just rubbed out poor Sam's memory, and being an honest little turtle, he will tell you that he doesn't know the meaning of the word THREE.FLAG.



When should you use ERPS? You should use ERPS if you want to start a new session with Logo. If you decide that you want to call something else by a procedure name that you have already used, you will either have to use ERPS or a statement like ER "THREE.FLAG. This will erase only the procedure THREE. FLAG. All of the other procedures that Sam knows about will be left untouched, if you use

ER followed by the name of the procedure.

Now that we know how to save a procedure and how to erase a procedure from Sam's memory, let's find out how to load a procedure into Sam's memory.

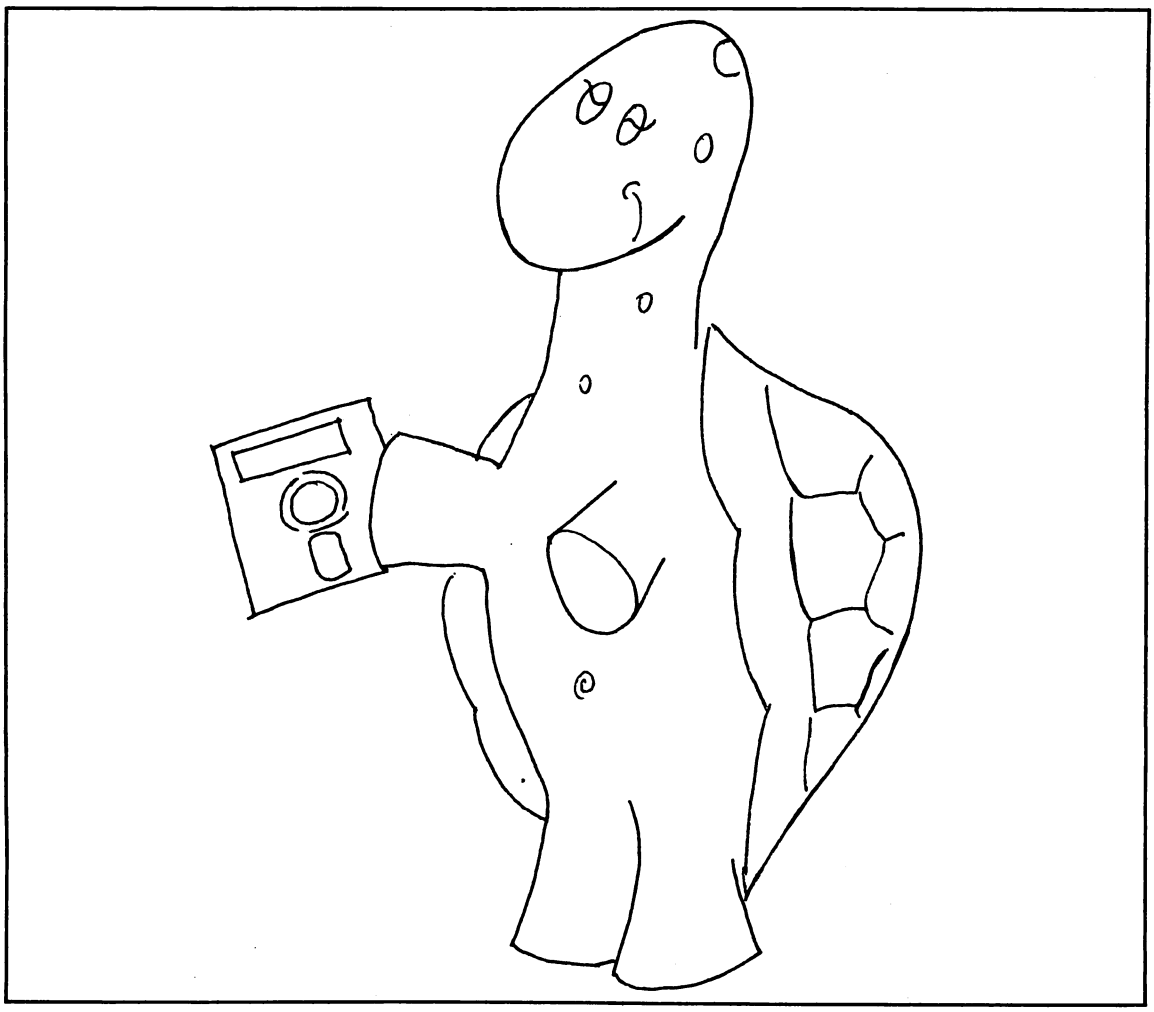
LOADING PROCEDURES FROM YOUR DISK

At this stage, Sam's memory should be empty—that is, you have typed the Logo word ERPS at the keyboard. Let us see if we can recall the THREE.FLAG procedure from disk! To do this, simply type the following:

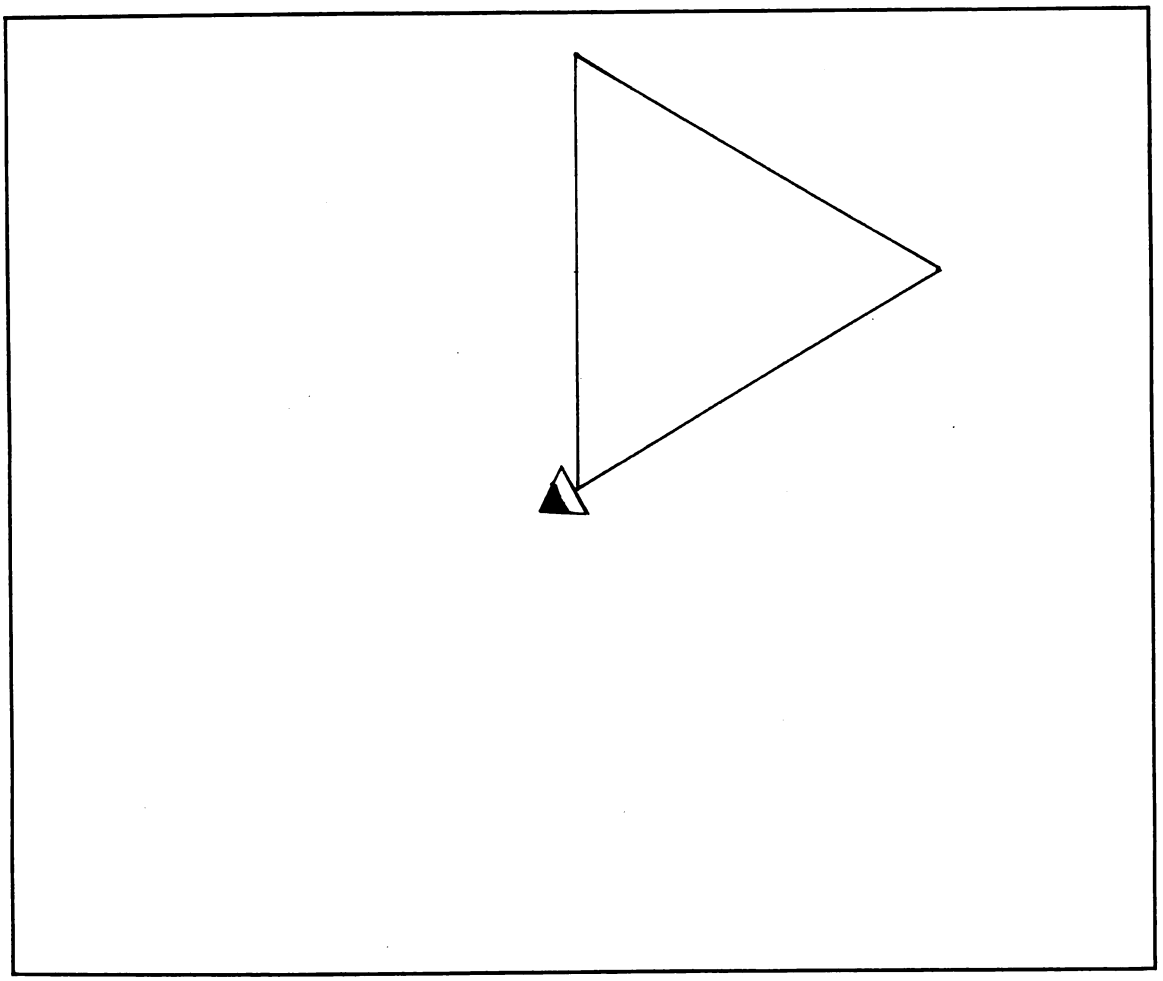
```
LOAD "THREE.FLAG
```

The quotation mark and the space are essential; be sure that you remember to type them in!

The disk drive will whir for a second or two as the procedure is loaded. When you see the question mark and cursor again, you will know that Sam is ready to hear from you once more. Type in the command THREE.FLAG to



Sam and his permanent memory!



Triangle.

make sure that everything is alright and that Sam does indeed know how to THREE.FLAG again.

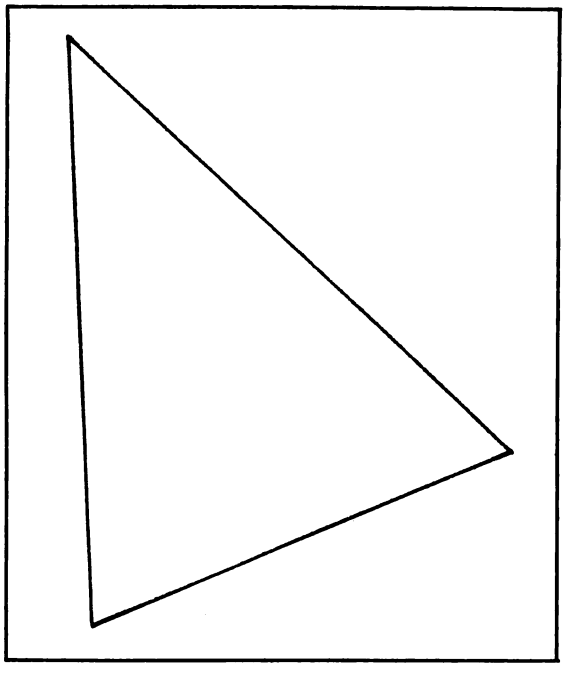
Earlier, we looked at how to teach Sam the meaning of the word SQUARE. I wonder if we can teach him the meaning of the word TRIANGLE?

TEACHING SAM HOW TO TRIANGLE

Now that we know how to teach turtle, let's see if we can teach him how to draw a triangle. You can think about this, and try it for yourself. It is fun to try. Try two or three times before you give up. If you find it difficult, ask yourself what the difficult part really is. Think about what Sam would do if he were in your shoes! You can read on after a while and look at one of the possible solutions given below.

Here is a procedure that will teach Sam the meaning of the word triangle. Type in the following.

```
TO TRIANGLE
FD 100
RT 120
FD 100
```



Triangle.

```
RT 120
FD 100
END
```

(If you did try to create a triangle by yourself, you perhaps found it hard to close the triangle's last side. The numbers that we used in our example were three 100's and three 120's. These numbers will make a special triangle! By using these numbers, we made our triangle close nicely.)

After you have typed in the procedure, type the following to see Sam draw the triangle.

```
TRIANGLE
```

Page 29 shows what you should see on the screen.

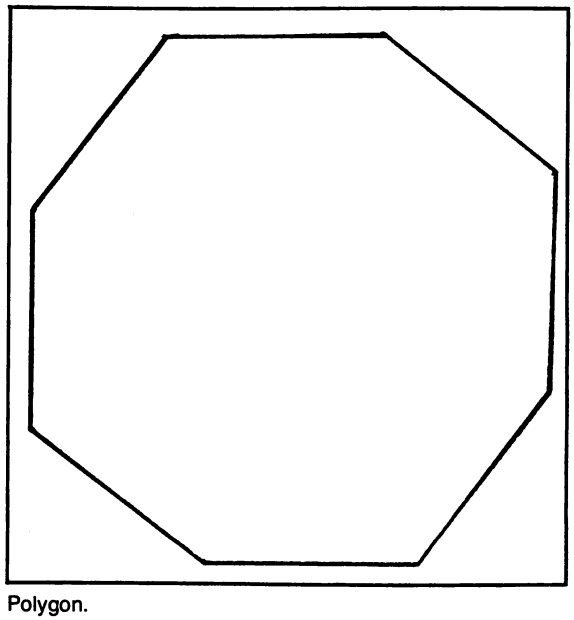
Teaching Sam how to triangle forced us to look at the numbers we used very carefully. This means that we learn a lot about geometry as we use Sam!

Here is a way to draw a triangle without having to think about the numbers that we need! Type in the following procedure.

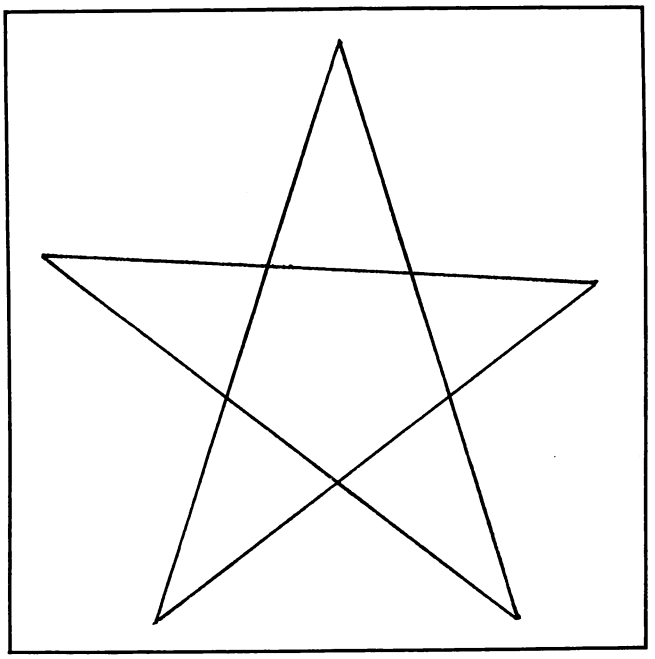
```
TO TRI
FD 100
RT 135
FD 90
HOME
END
```

What do you think made the difference here? None of the numbers are special, and the triangle isn't a special one either. What is new about the procedure? Of course, the procedure has a new primitive in it. The new primitive is the primitive HOME. HOME will send Sam to the middle of the screen, and it will make him face north. It is a nice way of closing a triangle, if Sam started off from the HOME position!

We can teach Sam about a lot more than just triangles, squares and flags. We can teach him all sorts of things. For instance, you can teach him how to draw a polygon that looks like this,



Polygon.



Star.

or a star that looks like this.

If you want, you can stop here and try out some of these ideas. Can you teach Sam how to draw these pictures?

Now that we know how to teach Sam how to draw a triangle, a square, and a flag, let's look at another powerful idea for computers that we can use in Logo.

MAKING SAM DO THINGS OVER AND OVER AGAIN

People don't like to have to do things over and over again. They really find it very tedious. Machines don't seem to mind doing tedious jobs at all! Machines are very patient. Sam, is a computer and a machine, and he is especially patient!

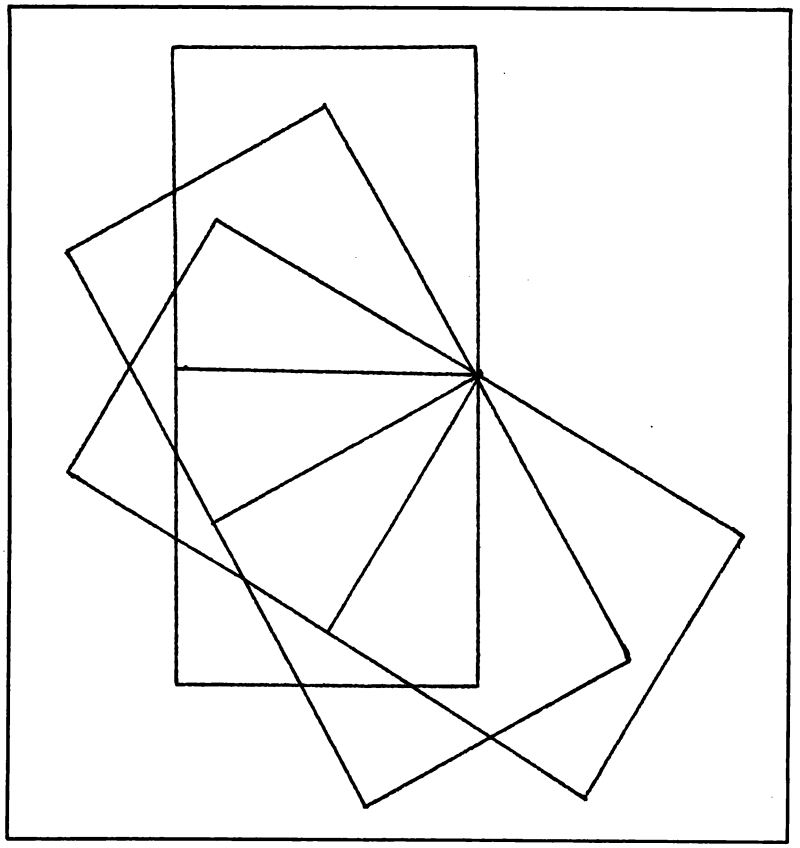
I wonder how we can get Sam to do things over and over again? Let's find out!

We can make Sam do things over and over again by simply typing them over and over. For instance, if we wanted to teach Sam to draw a pattern with the procedure square, we could type the following.

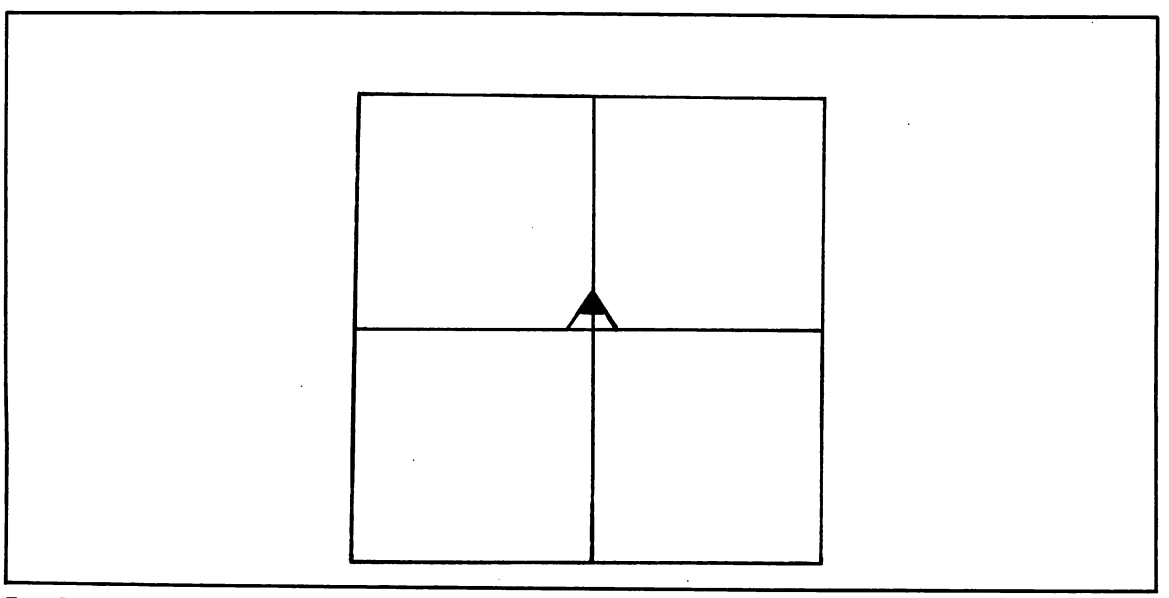
```
TO PATTERN
SQUARE
RT 60
SQUARE
RT 60
SQUARE
RT 60
SQUARE
RT 60
SQUARE
RT 60
SQUARE
RT 60
SQUARE
RT 60
END
```

To see the pattern that this will make, simply type the word PATTERN at the keyboard.

Of course, that was quite a lot of typing. I wonder if Logo has a better way of doing things over and over again?



PATTERN



Four Square.

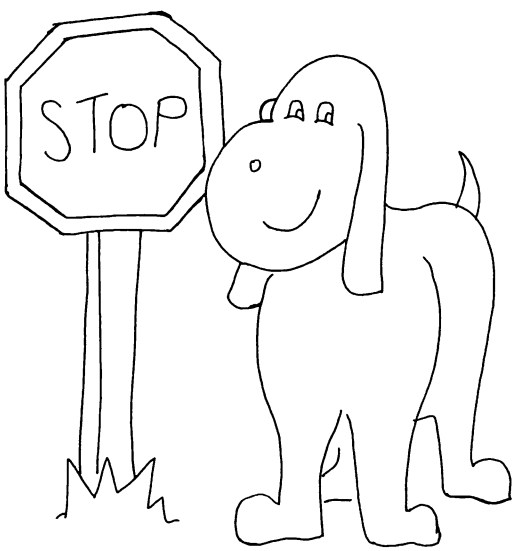
Making Sam Do Things Over and Over Again with Repeat

We will use the word *repetition* to describe the idea of something being done over and over. Let's use repetition with Sam.

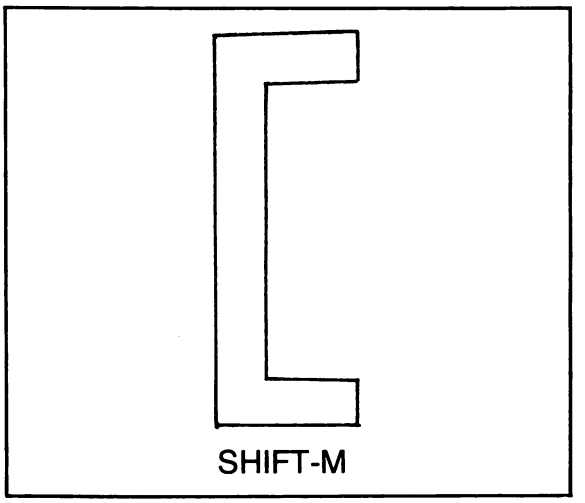
Make sure that Sam knows how to TRIANGLE and SQUARE. You can make sure that he knows by typing the word TRIANGLE and watching Sam draw a triangle. Then try typing the word SQUARE, and make sure that Sam will draw a square. If he doesn't know how to TRIANGLE, or SQUARE, then either load the procedures from disk or type in the TRIANGLE and SQUARE procedure once again at the keyboard. When you are sure that he can TRIANGLE and SQUARE, type in the following:

```
REPEAT 4 [SQUARE]
```

This will give you the pattern on page 32.



NOTE: TO MAKE THE BRACKETS SHOWN, USE SHIFT M FOR THE OPEN BRACKETS, AND SHIFT N FOR THE CLOSING BRACKETS. YOU *HOLD* DOWN THE SHIFT KEY, AND THEN PRESS THE M OR THE N KEY.

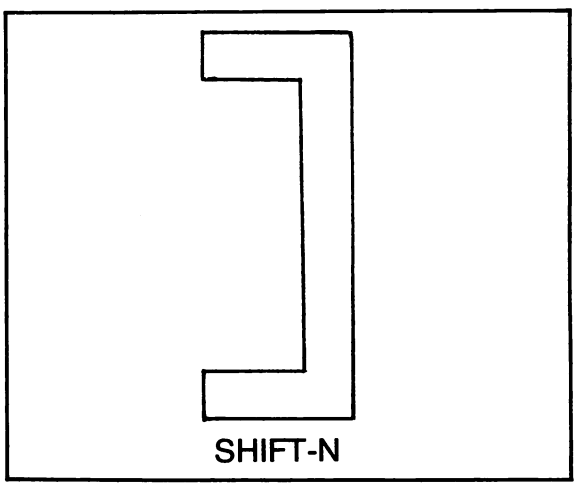


Open brackets.

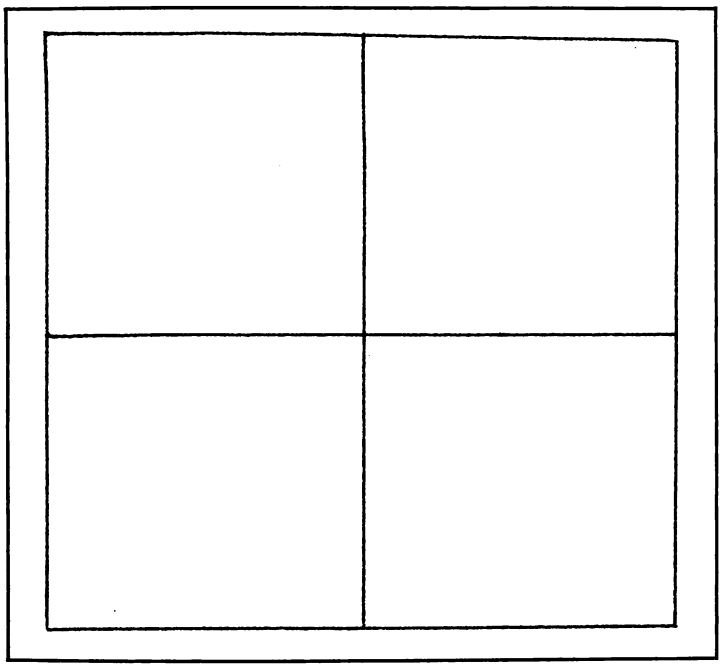
IF YOU ARE USING APPLE *IIe*, SIMPLY USE THE KEYS MARKED WITH [and] ON THE KEYBOARD.

Let's teach Sam to do something using the repeat command!

```
TO FOUR.SQUARE
REPEAT 4 [SQUARE]
END
```



Close brackets.



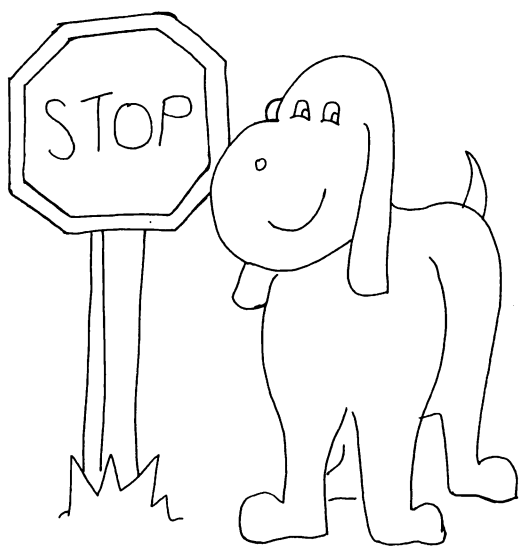
FOUR.SQUARE.

To run the procedure type FOUR.SQUARE.

Here is a procedure that will draw even more squares!

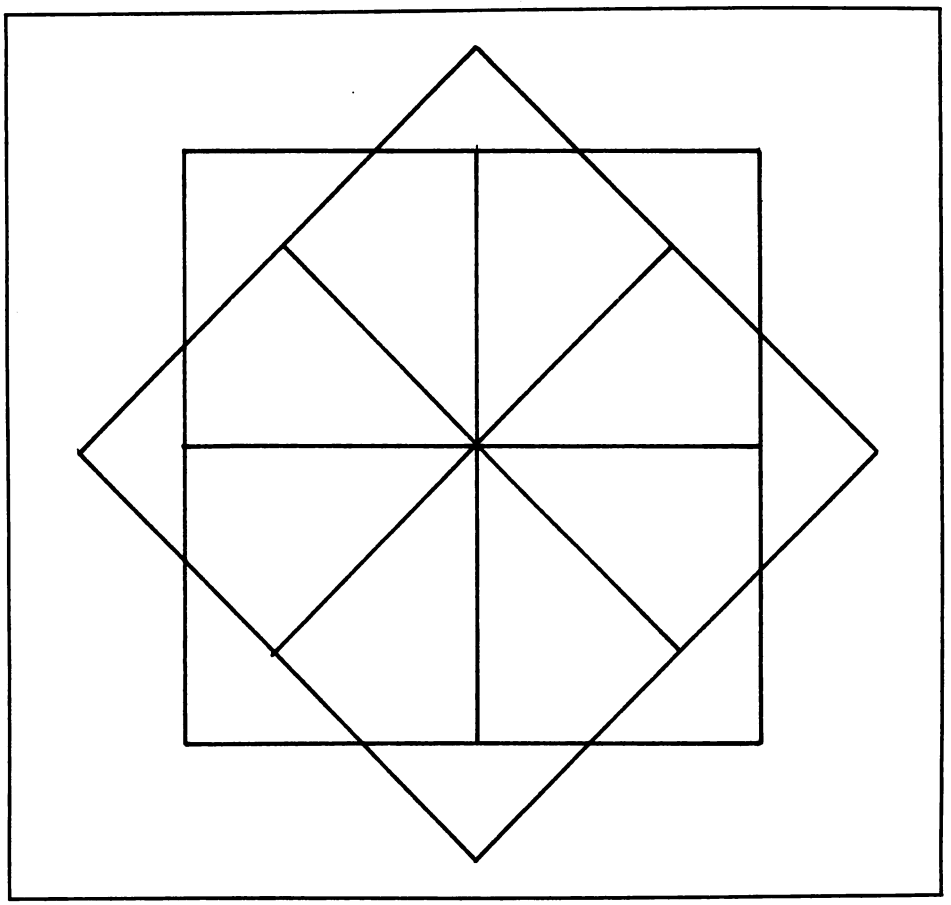
```
TO EIGHT.SQUARE
REPEAT 4 [SQUARE]
LT 45
REPEAT 4 [SQUARE]
END

TO SQUARE
FD 50
RT 90
FD 50
RT 90
FD 50
RT 90
FD 50
END
```

To see Sam draw the pattern, type EIGHT.SQUARE. Did you notice that we included the procedure SQUARE this time? You would have to type it in this way to make the procedure run after switching off your computer and starting Logo from the beginning again.



In the example, we typed in two procedures at the same time. We can type them in at

EIGHT.SQUARE.



the same time, or separately. This is very common in Logo; a procedure might use many subprocedures, and these in turn might call many others. You don't have to type them in at any special time. As long as they exist in Sam's memory when they are called, it will be alright!

If we had wanted to, we could have replaced the procedure SQUARE with the following.

```
TO SQUARE
REPEAT 4 [FD 100 RT 90]
END
```

This is a much shorter procedure than before. This is a good example of how to use the REPEAT command.

Here is another procedure for you to try; this time instead of using a square, we will use a long skinny rectangle.

```
TO WINDMILL
REPEAT 4 [SKINNY.RECTAN-
GLE RT 90]
END

TO SKINNY.RECTANGLE
FD 70
RT 90
FD 20
RT 90
FD 70
RT 90
FD 20
```

```
RT 90
END
```

To see the windmill, type WINDMILL and then press the return key. Here is what you will get!

Can you think of a better way of writing the SKINNY.RECTANGLE procedure. Let me see! How about this one.

```
TO SKINNY.RECTANGLE
REPEAT 2 [FD 70 RT 90 FD 20 RT
90]
END
```

That is much better! Use REPEAT a lot in your procedures.

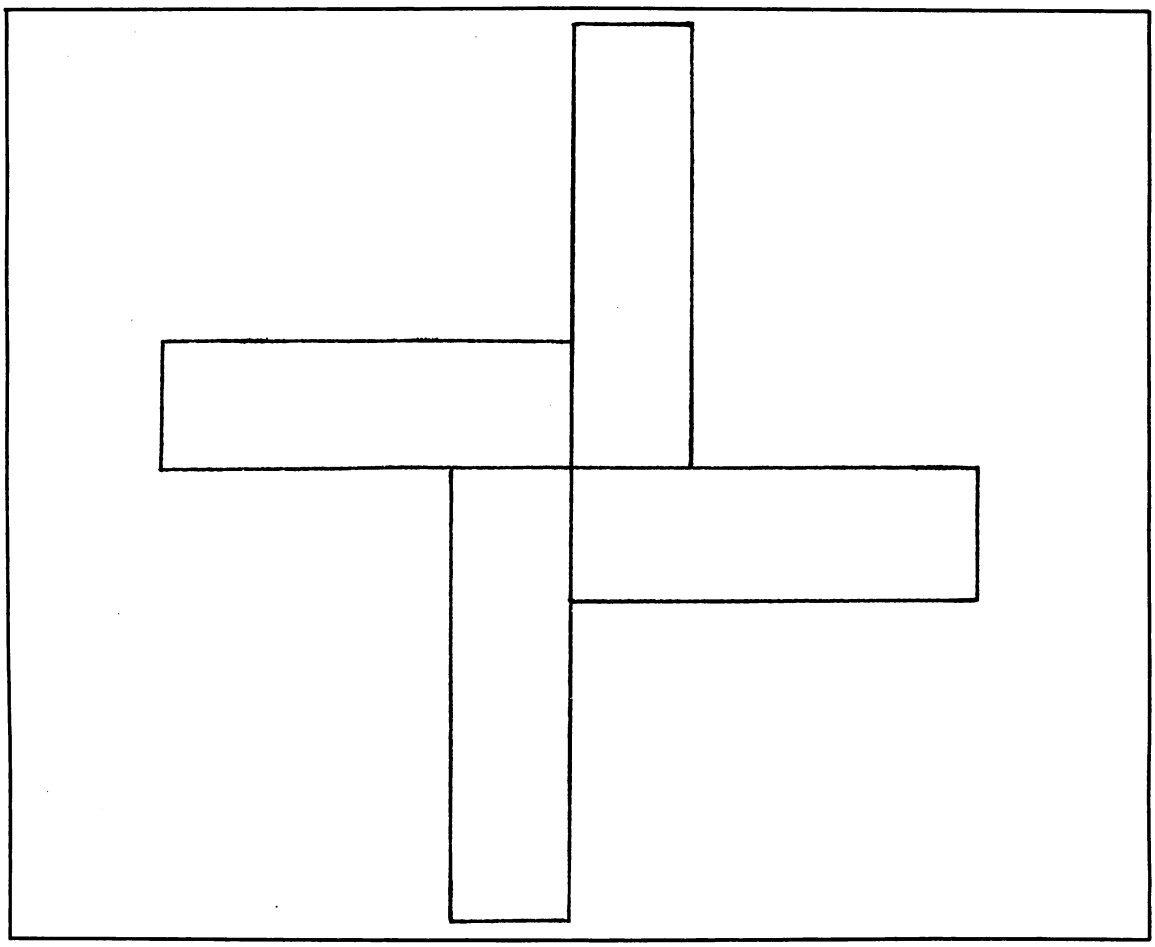
I wonder what other things we can draw?

MAKING A DANDELION

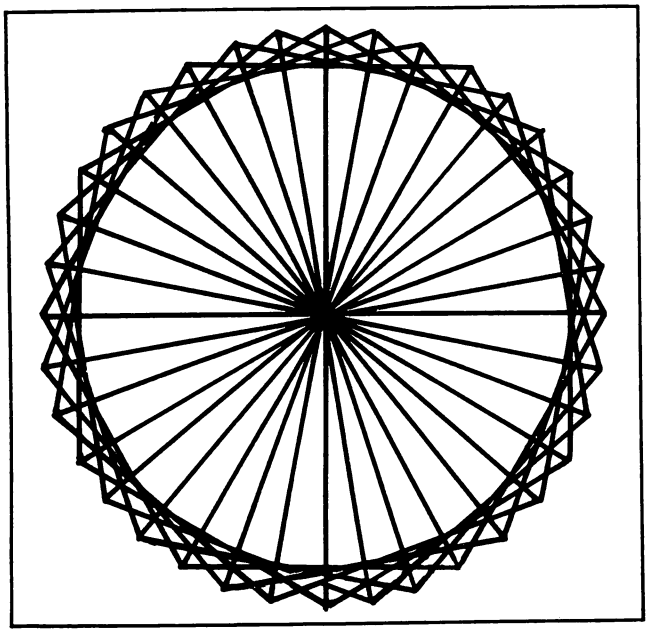
Here is another procedure for you to try. Make sure that Sam knows how to draw triangles, and then type in the following.

```
REPEAT 36 [TRIANGLE RT 10]
```

You should see the picture on page 37 on your screen: I suppose that this sort of looks like a



WINDMILL.



Dandelion.

dandelion, let's see if we can TEACH Sam the meaning of Dandelion. Type in the following:

```
TO DANDELION
REPEAT 36 [TRIANGLE RT 10]
END
```

When you have done this, type DANDELION to see Sam draw the dandelion.

Here is another example of repetition for you to try.

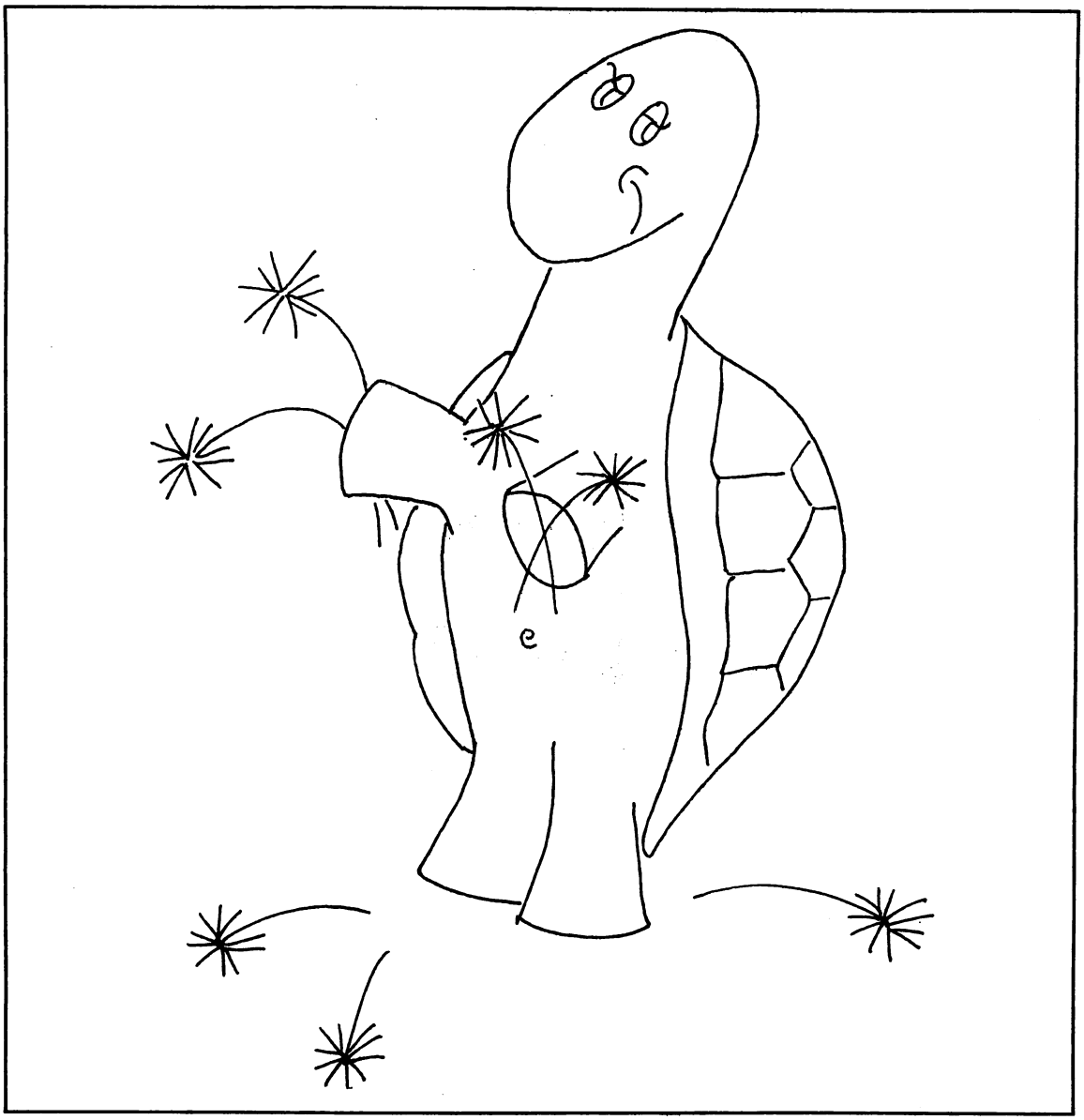
```
TO MANYLEG
REPEAT 18 [LEG RT 20]
END

TO LEG
FD 50
RT 90
FD 30
BK 30
LT 90
BK 50
END
```

Type LEG to see the procedure LEG run, then type MANYLEG to see the procedure using LEG over and over again.

SUMMARY

Read the following summary carefully, and then try some of the challenges that follow. There are also a quiz and a crossword puzzle at the end of the chapter.

1. A *procedure* in Logo is a group of Logo primitives written one after the other.
2. You can teach Sam how to do things by defining a procedure using the TO command.
3. You can save and load your procedures to and from disk. This means that you don't have to retype your procedures each time you switch on the machine.
4. ERPS is a command that will erase *all* of the procedures in Sam's memory.
5. One of the most powerful things a computer can do is to repeat things over and over again. This is called *repetition*. In



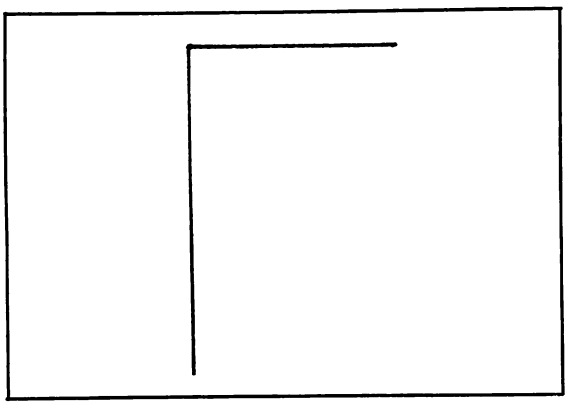
Sam likes dandelions!

Logo, repetition is done with the REPEAT command; for example

```
REPEAT 5 [LEG]
```

would draw the procedure LEG five times.

6. You can use repetition to draw, among other things, interesting patterns.
7. The HOME primitive will send the turtle back to his original position at the center of the screen facing due north.
8. One procedure can contain another procedure. When this happens, we say that the



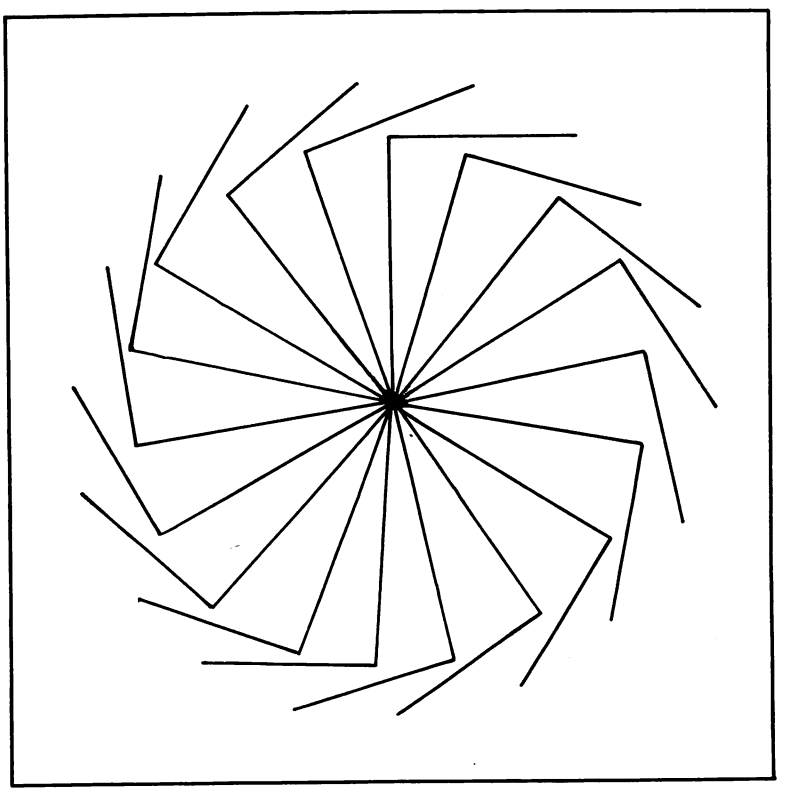
LEG.

first procedure *calls* the second procedure.

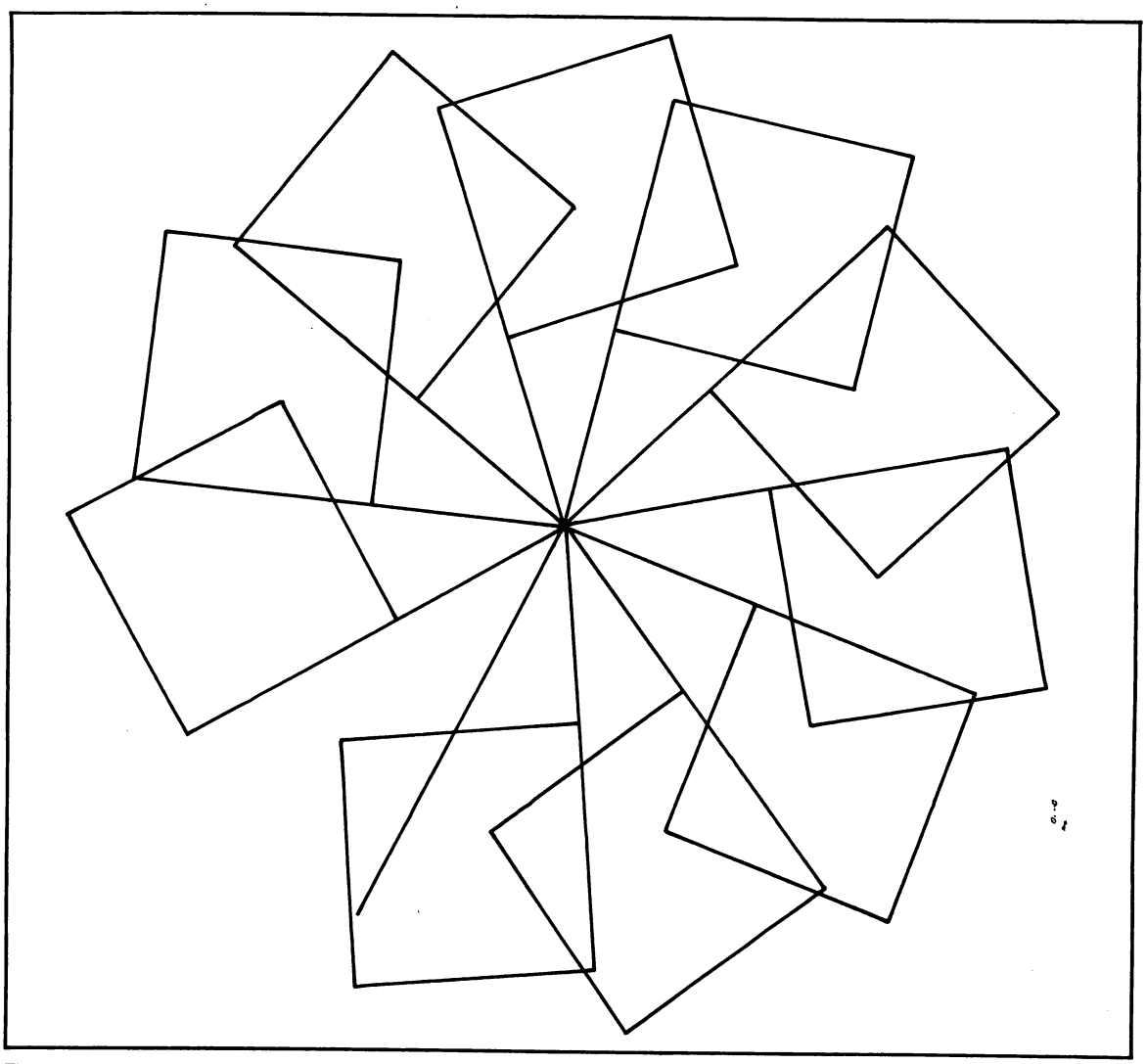
9. SHIFT-M and SHIFT-N are used to make the brackets used in the REPEAT command. If you have an Apple *II*e, then you should use the keys marked with [and].

CHALLENGES

1. Teach Sam to draw a rectangle.
2. Teach Sam to draw a hat.
3. Using REPEAT and the procedure HAT from Chapter 1, teach Sam to draw a hat pattern with many hats.
4. Draw an interesting diagram using REPEAT, SQUARE and RT.
5. Draw an interesting diagram using REPEAT, FLAG and LT.
6. Teach Sam to draw a pattern of horizontal lines.
7. Teach Sam to draw the following flag pattern shown on page 40.
8. Teach Sam to draw a zigzag pattern. You can use the diagram shown on page 41 as a guide if you like.
9. Teach Sam to draw a kite like the one shown on page 42.



MANYLEG.



Flags full circle.

QUIZ

1. A procedure in Logo is a group of Logo _____ written together.
2. You can teach Sam how to do things by defining a procedure using the _____ command.
3. You can save and _____ your procedures to disk. This means that you don't have to _____ your procedures each time you switch on the machine.
4. ERPS is a command that will erase _____ of the procedures in Sam's memory.
5. One of the most powerful things about a computer is that it can do things over and over again. This is called _____. This is done by using the _____ primitive in Logo.

REPEAT 5 [LEG]

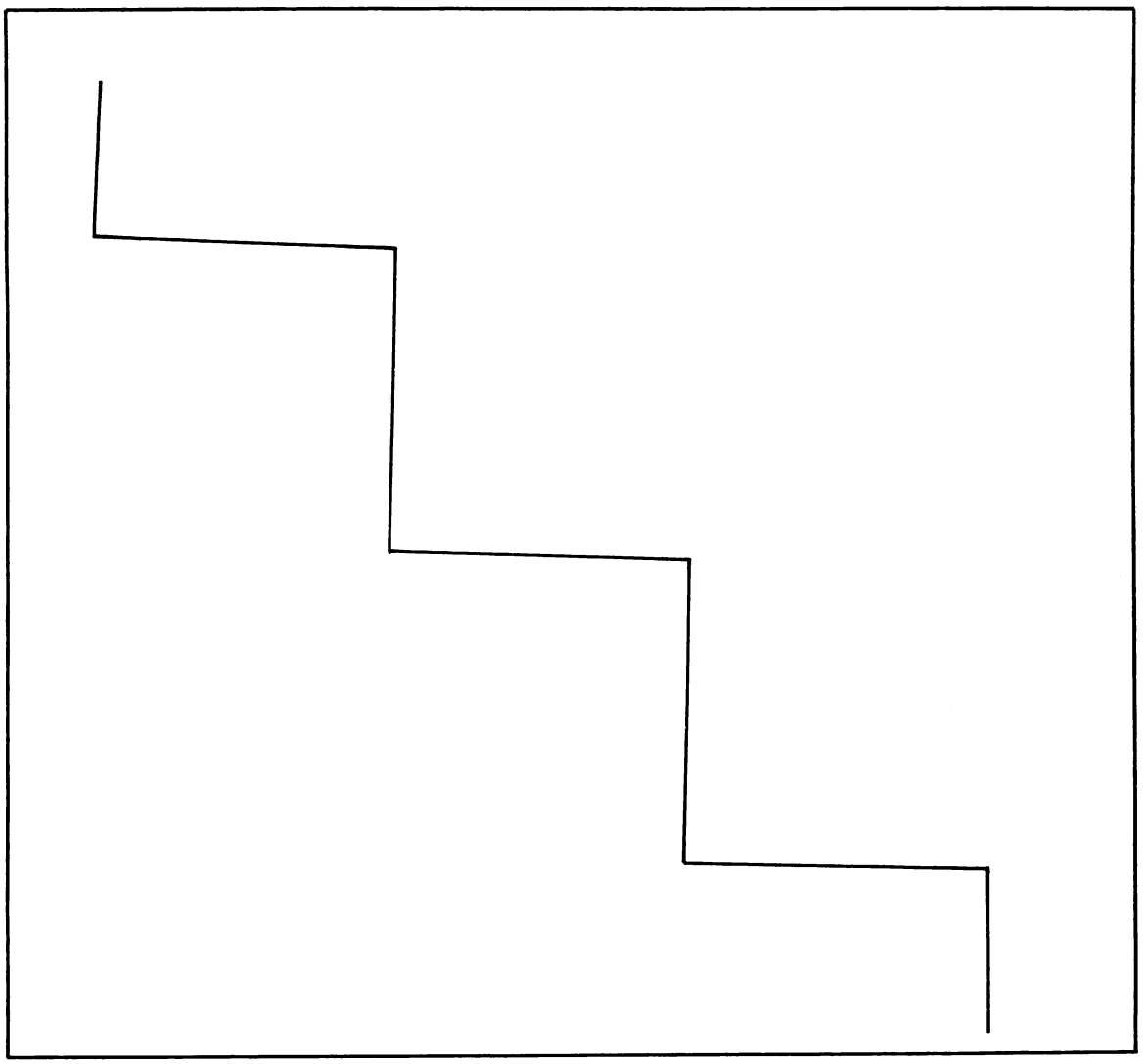
would draw procedure LEG _____ times over.

6. The primitive _____ will send the turtle back to his original position at the center of the screen facing due north.
7. One procedure can contain another procedure. When this happens, we say that the first procedure _____ the second procedure.
8. SHIFT-M and _____ are used to make the brackets used in the REPEAT command. If you have an Apple *IIe*, then you use the keys marked with _____ and].

CROSSWORD PUZZLE

Clues Across

1. This command will let you see the proce-

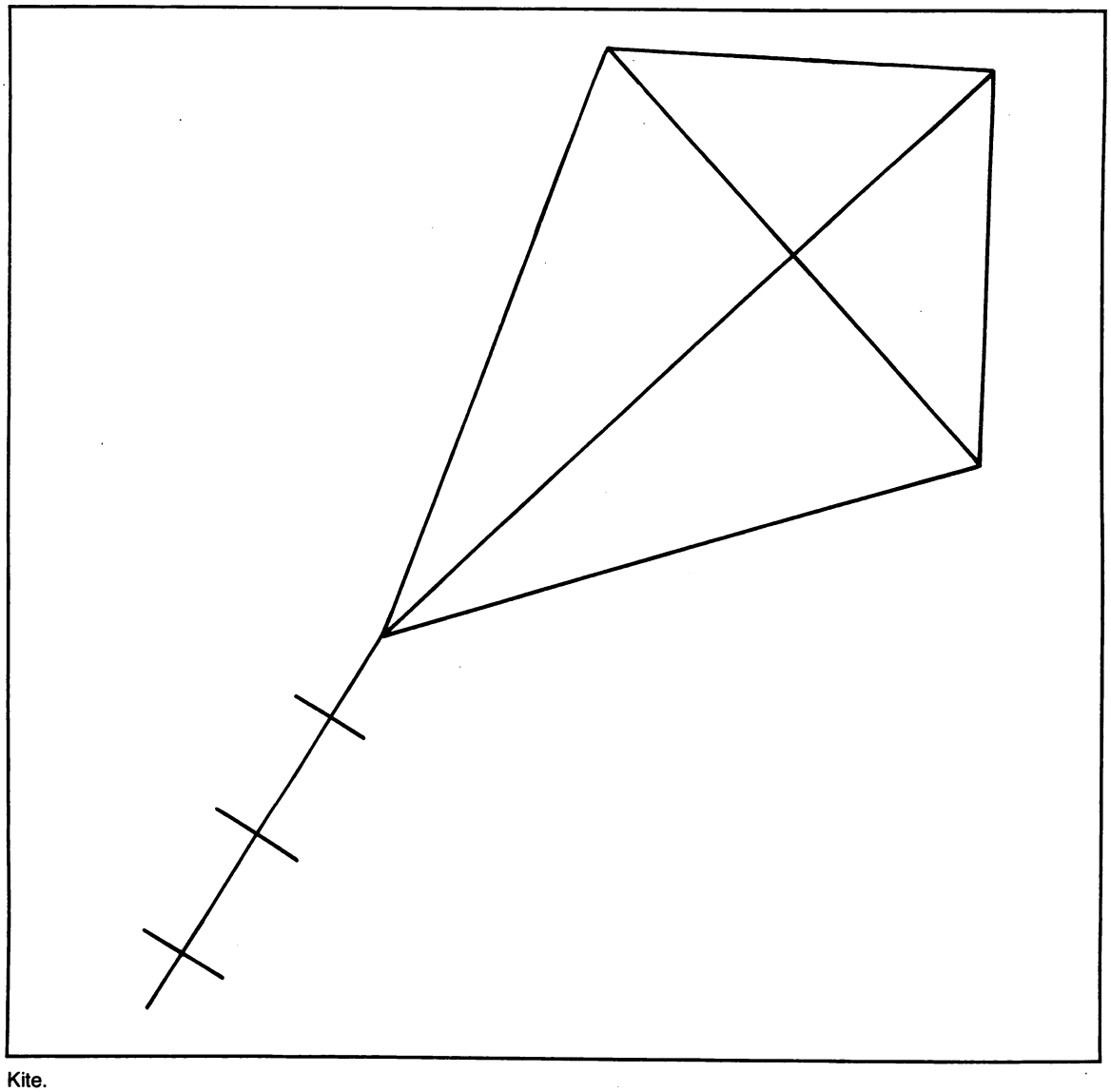


Zig-zag.

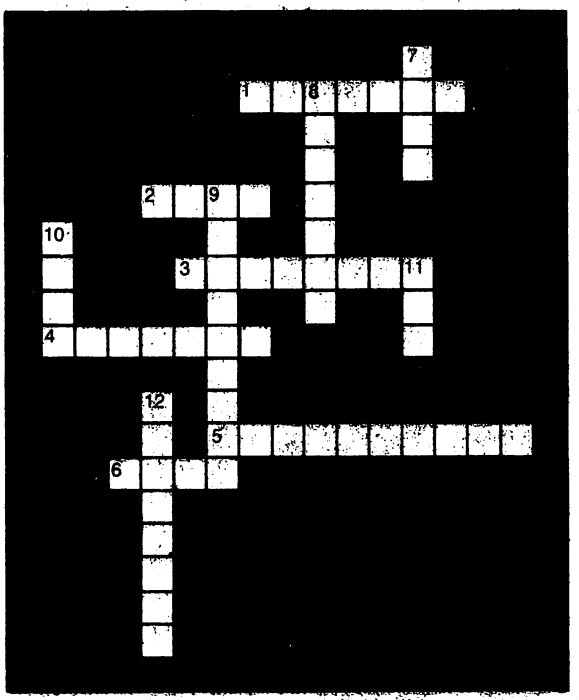
dures that have been saved on your disk.
2. If you want to rub out the procedures in Sam's memory, then use this command.
3. This will begin to teach Sam the meaning of the word SQUARE.
4. When you type END for 3 across, this will be part of Sam's reply to you.
5. I suppose that I will have to do this over and over and over!
6. This is what you type to send a procedure to your disk.

Clues Down

7. If you want to send Sam back to where he started, type this word.
8. When you want the turtle to do something, you _____.

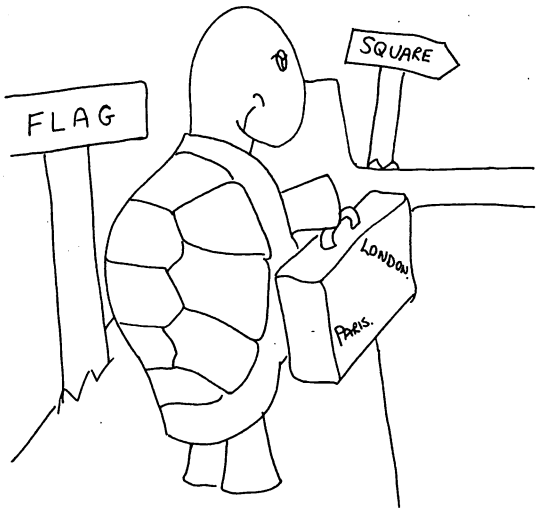


Kite.



9. This is made up of a list of primitives.
10. If you want to get your procedure from your disk into Sam's memory, use this command.
11. You always type this to finish off a procedure.
12. With Logo, we can _____ the meaning of words.

Chapter 3



What To Do If You Are Stuck

This chapter is intended to help you with problems and problem solving. You have a choice of reading this chapter, and then continuing on to the next chapter, or of skipping this chapter for the moment and then returning to it when you find that you have the need for advice on problem solving. The third possibility is that you read this chapter quickly and then go on to the next chapter.

This third choice is probably the best. This chapter is here, in this position, to let you know that there is a chapter that will help you in problem solution. Logo, like all computer languages is designed to solve problems, it is important to have some knowledge of the techniques that people use to solve these problems.

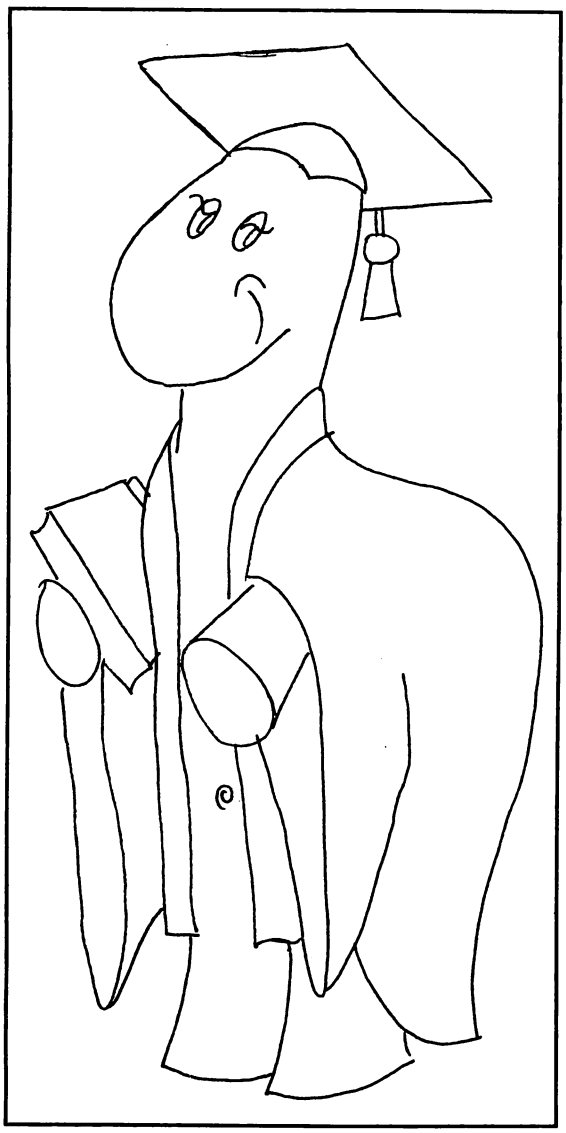
Although our turtle friend can follow instructions and can even learn the meaning of procedure names like SQUARE and TRIANGLE, he is really not that smart. He lacks what is called *intuition*. Intuition is the ability to go straight to the solution of a problem without working through all of the possible solutions. Sam can't solve problems in the way that you and I can!

For instance, if you were riding in the car with your dad and you found that you had a flat tire, your dad might immediately say "Oh dear I left the spare in the garage by mistake!". A computer would have to say "Lets check the jack; lets check the tire wrench. Lets check the spare wheel . . . . The spare wheel is missing!". Sam and computers examine problems in a step by step fashion. They cannot suddenly produce direct results like we sometimes can. Computers are not able to solve problems by using intuition or imagination.

Because you can solve problems by using your experience and imagination and by trial and error, you are going to have to help Sam to solve his problems. You will find that Sam will

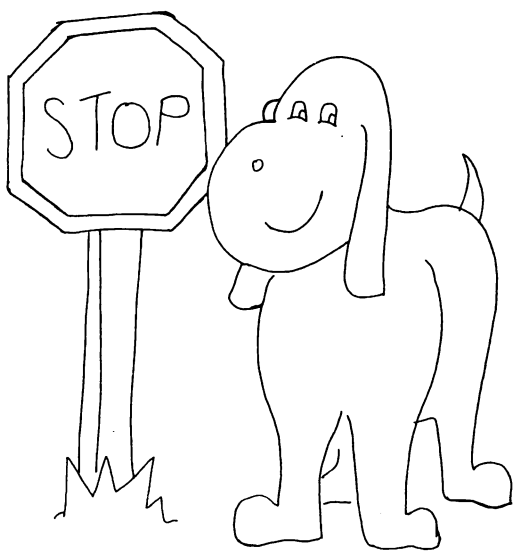
do exactly as you tell him! If you tell him to do the wrong thing, then he will do the wrong thing! Computers are like that!

What is the wrong thing? In Logo, there are no wrong things! This is because, an error in a procedure will often lead you to new ideas, as well as possible solutions to the problem that you are looking at. If we use our errors wisely, Sam will teach us many things. He has already taught us about simple geometry and programming. Perhaps we should have called Sam a tortoise, because he has taught us so much!



Perhaps we should have called Sam a tortoise because he has taught us so much.

In Logo, you learn from your errors! However, sometimes you will ask Sam to do something that will give you results you did not count on. For instance, suppose that you wanted to add a roof to a square house, and it turned out that the roof appeared on the side or on the bottom of the house instead of on the top. You might consider that as a mistake or a *bug*. When we fix errors in a program, we talk about *debugging* the program. If we had our roof on the wrong way for instance, we could fix the bug by rewriting the procedure in such a way that the roof appeared on top of the house. The important thing to remember about bugs is that you will learn a lot as you try to remove them from your procedures.



What *is* right and wrong in Logo? The answer to this question isn't black or white. If you enjoy using Logo and you enjoy telling Sam to do things then you are being successful with

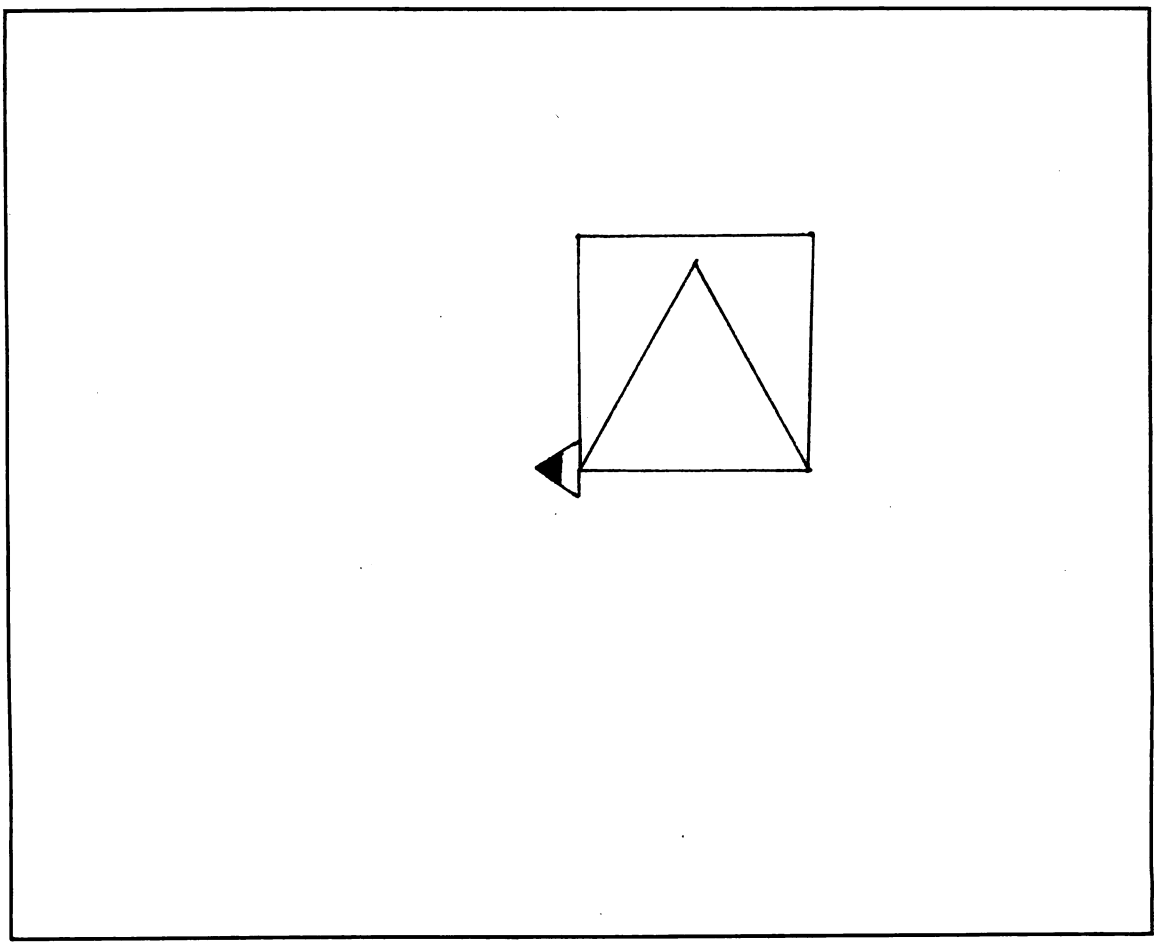
the language. The important thing about your mistakes in Logo is that they lead you to something new concerning the problem or something new in the language. In short, you learn from your mistakes.

In the chapters that follow, you will find a variety of suggestions and problems for you to try. None of the problems are really brainteasers. That isn't what we intend to look at this time round. The suggestions will help you to learn about Logo and will show you how to think about doing things.

If for instance the challenge is to draw a picture of a witches hat, you might find that your first attempt looks a lot like a space shuttle. You should feel free to work on your picture of the shuttle. This is what Logo is about; it gives you the freedom to experiment with ideas and problem solving.

Sam loves to obey you and display your ideas. It is fine with Sam if you decide to change your challenge into another pattern or picture.

Logo is intended to be fun, but it won't be long before you will find that sometimes, it seems that Sam will not do what you want him to do. Sam really is doing what he was told! Sam always does exactly what you tell him to.



Oops . . . . .wrong house!

After all, Sam is really only a collection of electronic devices in the computer. What is happening is that you haven't quite been able to give Sam the correct list of Logo instructions.

Perhaps you have left something out, or perhaps you have put something in that was not needed. Or maybe, there is something out of order in the primitives that you have used. This happens all the time in computer programming. When this happens, it is called, funnily enough, a *problem*.

Just to illustrate what a problem is, let us look at the example of drawing a house. We might have typed in the following, hoping that it would draw a house.

```
TO HOUSE
SQUARE
TRIANGLE
END

TO SQUARE
REPEAT 4 [FD 50 RT 90]
END

TO TRIANGLE
RT 30
REPEAT 3 [FD 50 RT 120]
END
```

When you type HOUSE you will find that you will get the picture shown on page 46.

What happened here, was that the roof appeared in the wrong place. This is typical of the type of problem that you might find with Logo and Sam.

Here are some ideas on how to solve this and some other problems. The thing about problems is that they are exactly that, problems. They are not easy to cope with or to solve. The hints that follow will be general. They will apply to all types of problems.

COMMON PROBLEMS THAT CONFUSE SAM

There are really two types of problems that you can have. They are

- Technical and language problems.
- How-to-do-it problems.

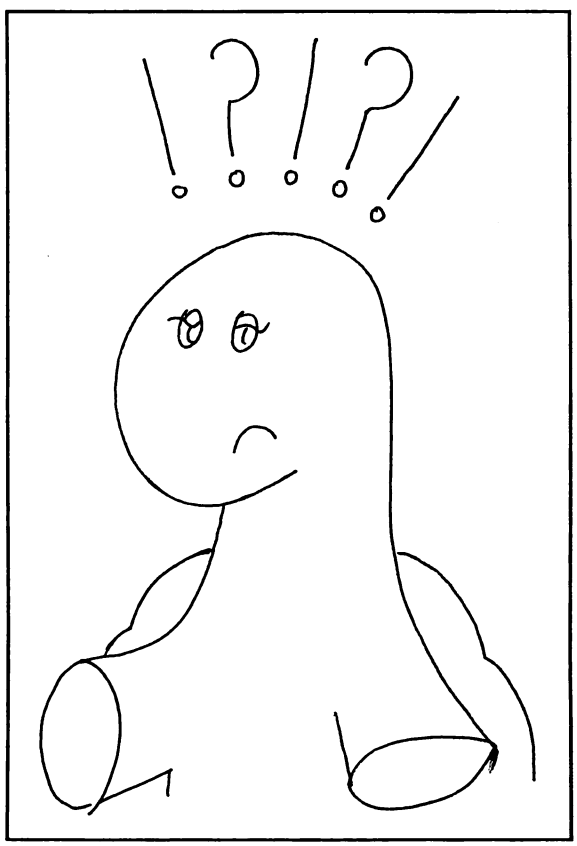
The technical problems are problems that confuse Sam. The "how to do it" problems are problems that confuse you.

Technical and Language Problems

This is the type of problem that you get when you type in things like

RT90 instead of RT 90.

Remember that you need a space between the RT and the 90!



Sam is confused!!

In computer jargon this is called a *syntax* error. This means that you have made either a spelling mistake, that you have left out a space as in the example above, or that you have forgotten to use a " mark where you needed one.

This type of error will show up immediately if you are in the *top-level.* You are in the top-level when you see the Logo question mark and cursor. You will be told later about the error if you have written the error into a procedure. It will show up when you try to call that procedure.

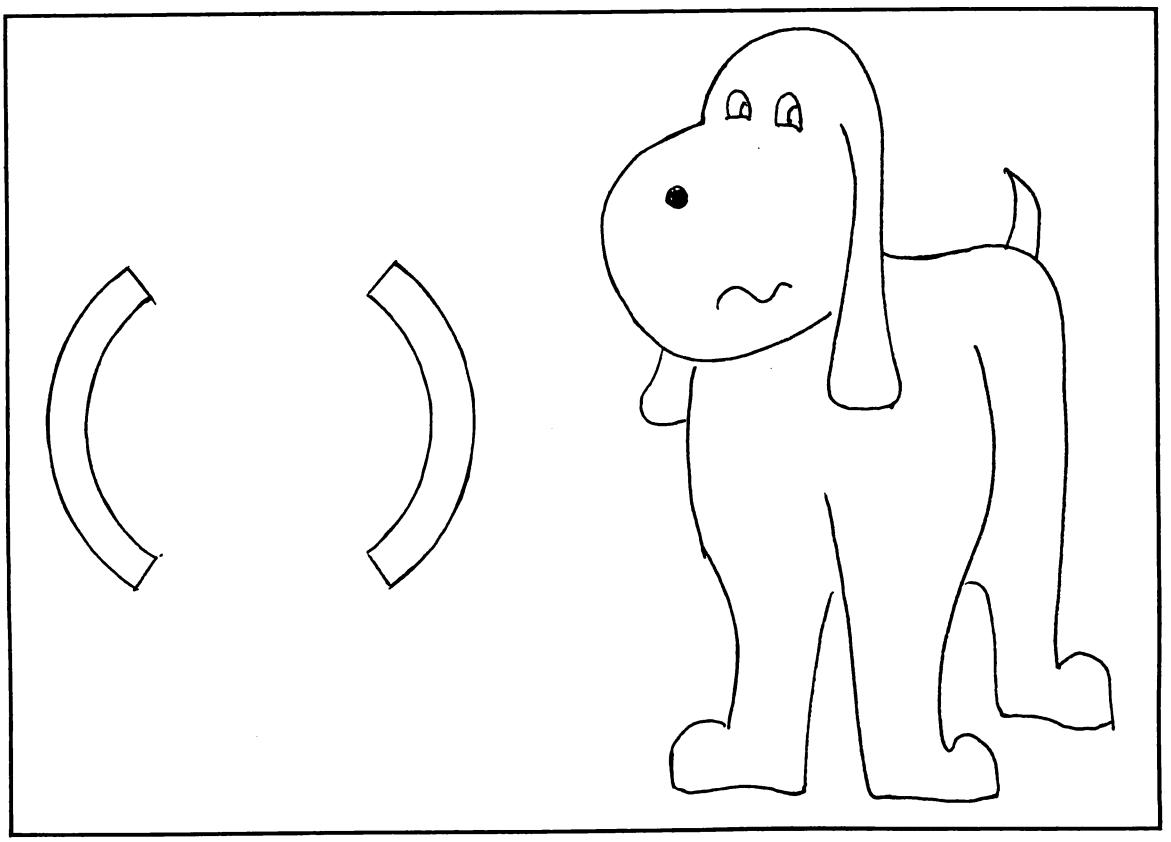
Here are some of the more common syntax problems that can crop up.

- Leaving out spaces that are required.
- Leaving out quotation marks.
- Using the wrong brackets in primitives such as REPEAT. The brackets required are generated by using the SHIFT-M and the SHIFT-N keys!
- Leaving out numbers when they are needed. For example, FD must be followed by a number.
- Misspelling primitive words, for instance, RPEAT instead of REPEAT.

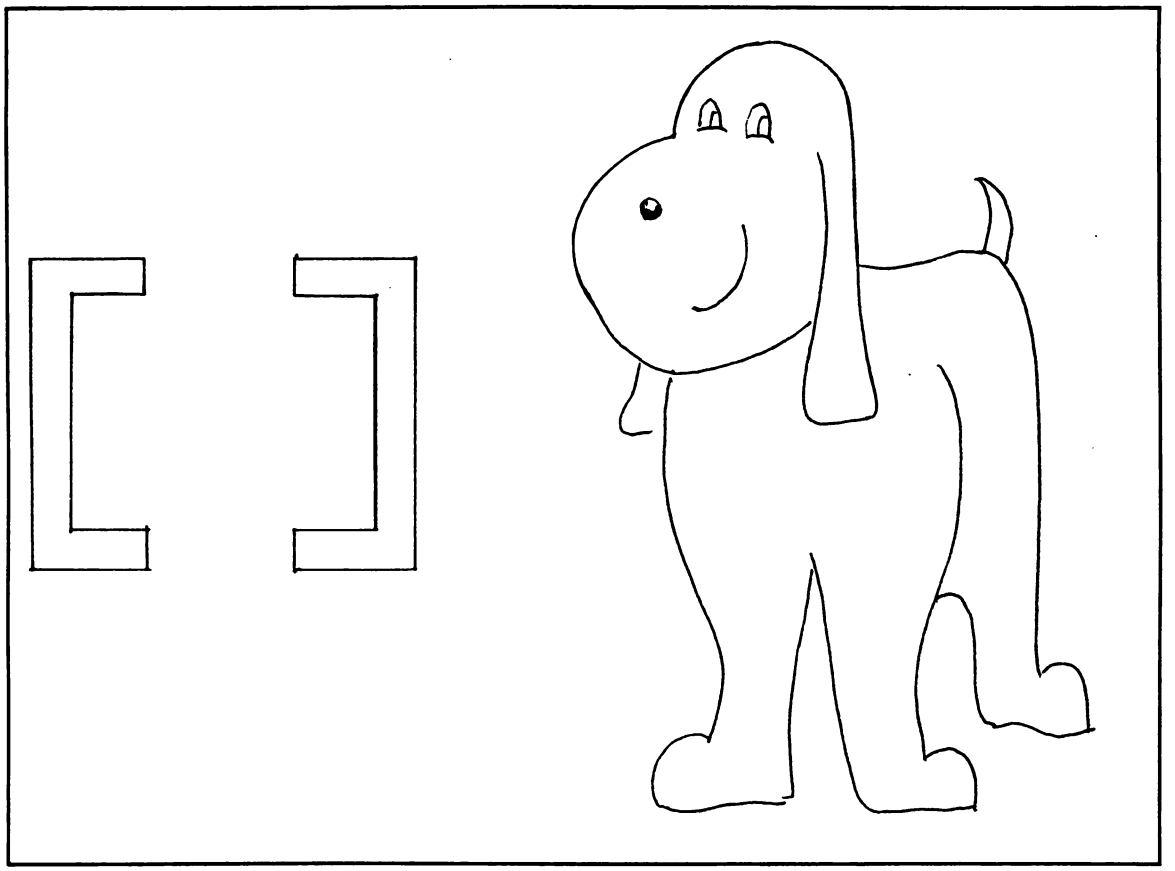
If you have a question about the syntax of a primitive, refer to the Apple Logo manual. Look at the examples that the manual gives, and compare the examples to your procedure.

PROBLEMS THAT CONFUSE YOU

These are the how-do-you-do-that?



Watchout for the wrong brackets!



Correct brackets.

problems, and they are usually the most difficult to solve. Here is a list of well tried methods that are known to work.

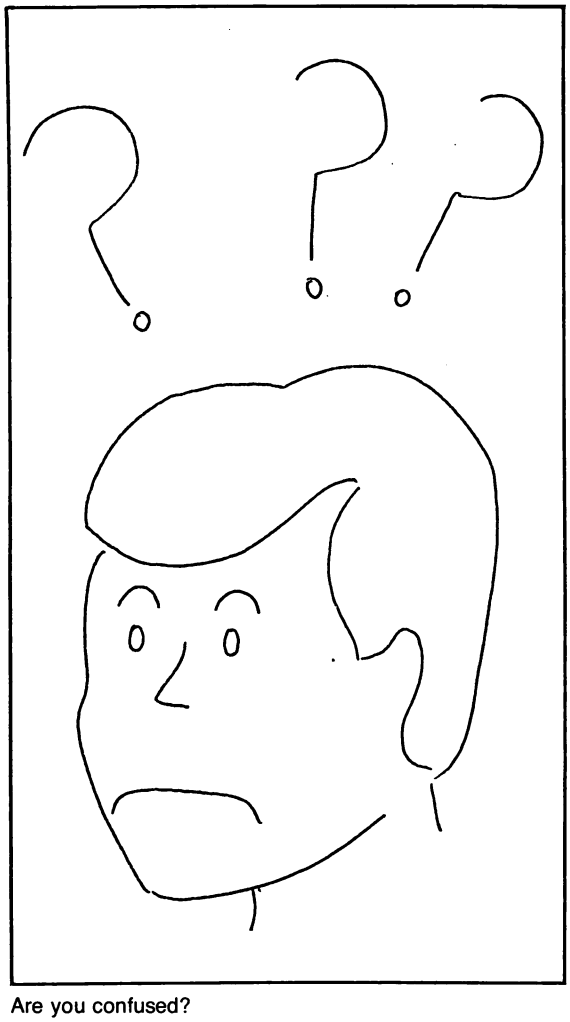
Break the Problem into Smaller More Manageable Steps. For example, let's suppose that you wanted to draw a house with windows. The first step would be to realize that a simple house is really a rectangle with a triangle on top of it. The windows and doors would be added last of all, after the outline of the house with the roof on had been drawn.

As another example, suppose that you wanted to draw a spaceship. The first step might be to draw the body of the ship, then the nose cone and finally, the two side fins. The body would be a long thin rectangle, and the nose cone and the fins could be made up of triangles.

When you finally write the procedure, make sure that each piece of your ship is written into separate procedures. You should have a procedure for drawing the body called BODY, a procedure for drawing the nose cone called NOSE.CONE and so on.

Look for a Similar Problem That Has a Well Known Solution. Then try to apply that solution to your problem with some suitable changes. The difficulty here is in determining which problems are similar.

For example you might want to draw a rose instead of the dandelion that we drew in the last chapter. Study the way that the dan-



Are you confused?

delion was drawn and then ask yourself "what is the same about the two problems and what is different?" If you wanted a rose instead of a dandelion, you might decide that you wanted a different shape of triangle, (the petals were made up of triangles remember!). You might also decide that really, you wanted fewer petals, because, a dandelion is fuzzier than a rose!

To make the right decisions, it is helpful to look at both the dandelion and the rose together. Compare the two illustrations on the next page to check the differences.

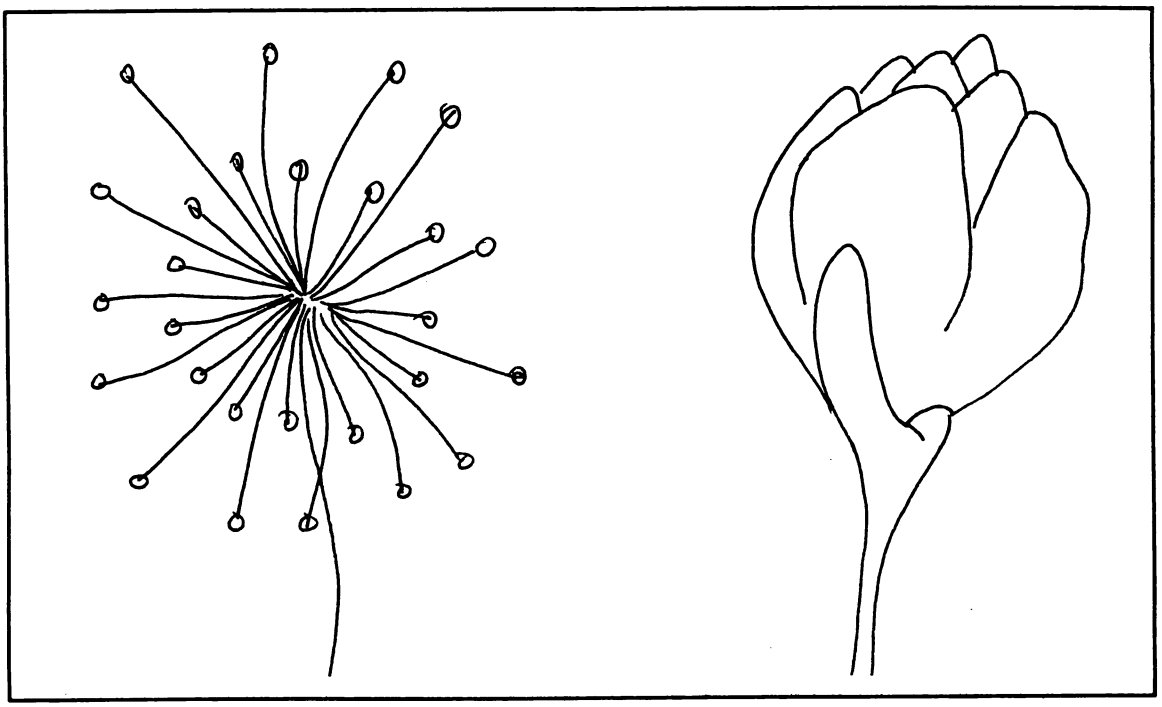
We would need fewer petals made of bigger triangles, but the stems would be the same, and the solution to the rose problem should be similar to the solution to the dandelion problem! Having made this decision, it is time to look at the procedure for the dandelion, and then make some changes to that procedure. When we run the procedure after making these changes, we should be prepared to make some more changes, it is unlikely that we would get the desired results after only one attempt!

Use Trial and Error. This is when you simply try an idea for a solution at random. For example, you could draw that house by starting at the keyboard and telling Sam to go forward, right, left and so on. If you didn't like the result, you could try again. If that didn't quite look right, then you could change something else a little bit, until you finally had the solution.

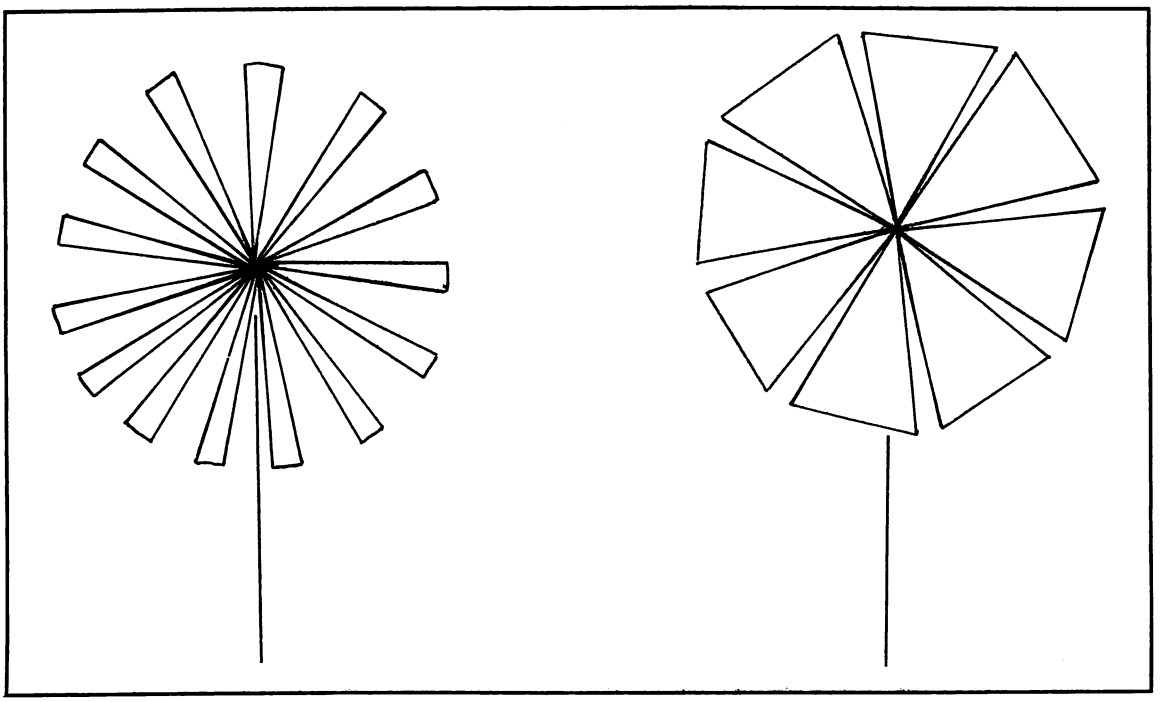
The trouble with trial and error as a method is that it takes a long time and it is easy to forget which parts of the trials were useful and which parts were not so useful.

Work from the Answer Backwards. This method is particularly good for telling Sam what to do, as most of the time, you will have a picture in your mind of what you want. Suppose that you wanted to draw a tree. When you begin, you probably know what type of tree you want. From this you could decide on how Sam would need to move in order to draw the tree. Often in Logo, if you are trying to meet a challenge, you will be able to look at the final result and then work back to the solution.

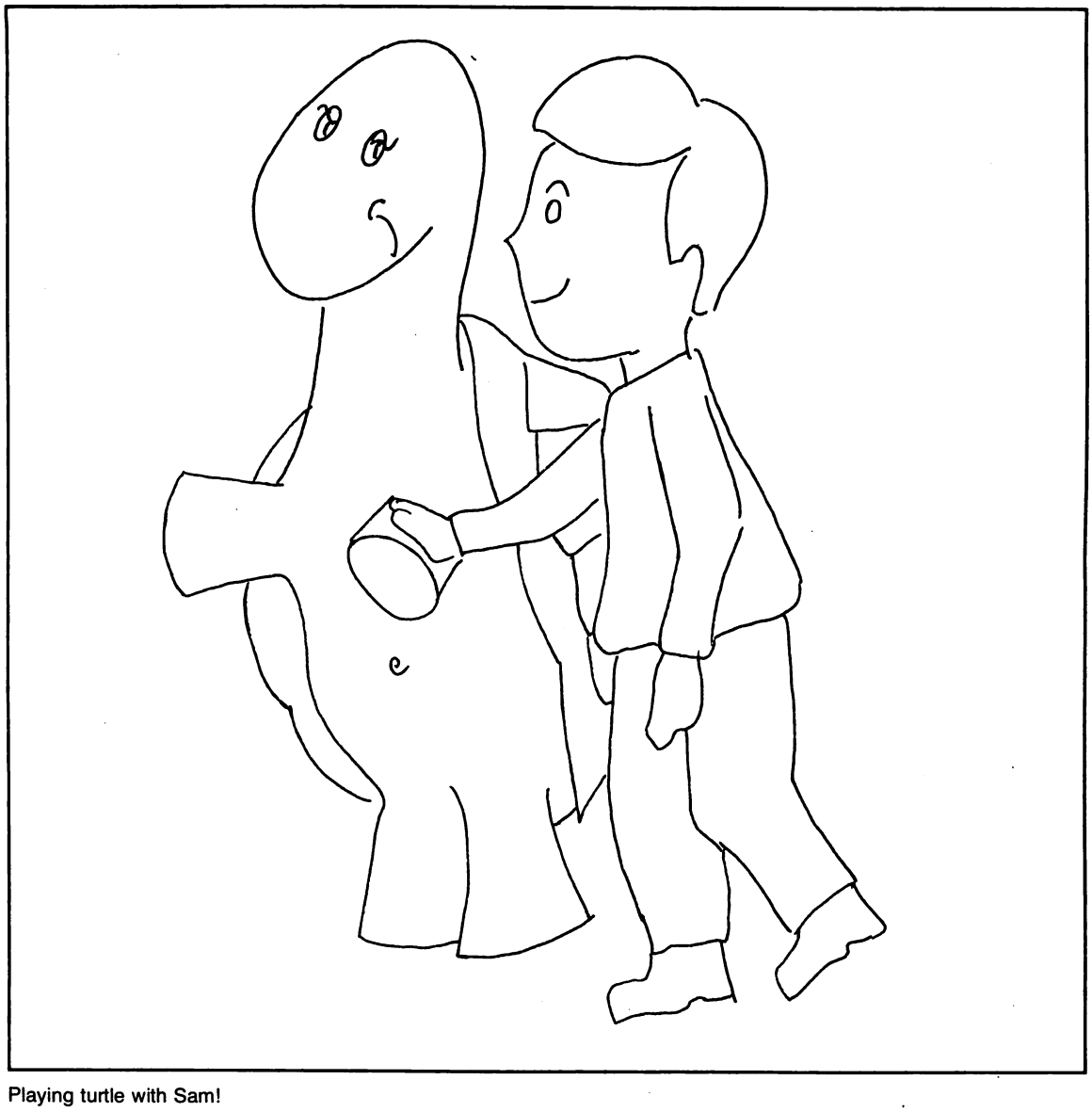
Work with a Friend Towards a Solution. This method is particularly good if you can find a friend with the same interests. You will find that this method is fun, as well as a very efficient way to find a solution to a problem. A group can often find a solution to a problem much faster than an individual can. In fact most advances in science and most large



A rose and a dandelion.



A rose and a dandelion made from triangles.



Playing turtle with Sam!

computer programs are created by teams or groups as opposed to individuals.

One method of problem solving that is not common in other computer languages is to pretend that you are in fact the program! Logo is particularly good at allowing you to pretend that you are the computer. Here is how to pretend that you are the computer!

PLAYING AT BEING SAM

One of the unique things about our turtle Sam is that you can pretend that you are Sam. If you run into a problem, you can simply stand in the middle of the room and pretend that you are the turtle. The room represents the screen, and you walk around the room obeying the primitives that you used in your procedure.

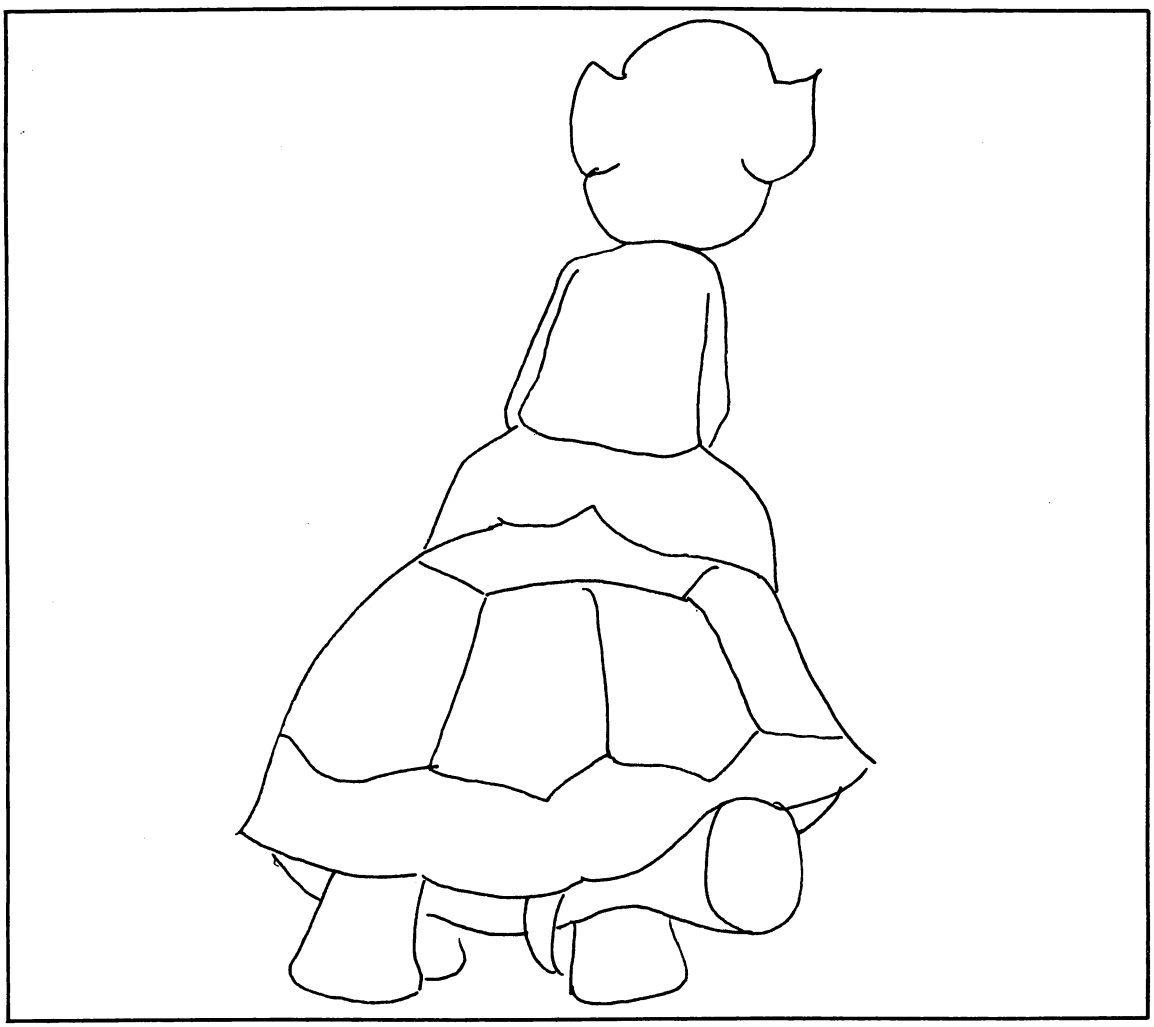
You can carry out the FD 10 command by taking 10 steps forward. The RT 90 command would mean turning to your right and facing the wall on your right.

When you pretend that you are Sam, we say that you are *playing turtle*. When you play turtle, you walk around doing the things that Sam has done for you so far. When you get to the part of your procedure that is causing the problem, you ask yourself "What should Sam do next? Should he go right, left, back or what? When you have decided on what you think he should do, you then take that result and write it into your procedure.

Remember that if you decide that you need to turn left, you also need to know how far to the left. That is, what value do you need to follow the primitive LT?

You will need to experiment a little to determine how much to move and so on. If you think that you need to move forward 30 steps say, try out the 30 steps before you decide finally on the best number.

Of course, a room doesn't have any wrap-



Showing Sam the way!

around, so please be careful not to bump into the walls! It is also sometimes difficult to simulate the REPEAT primitive by playing turtle, but all in all this a very powerful way of figuring out what you should do next!

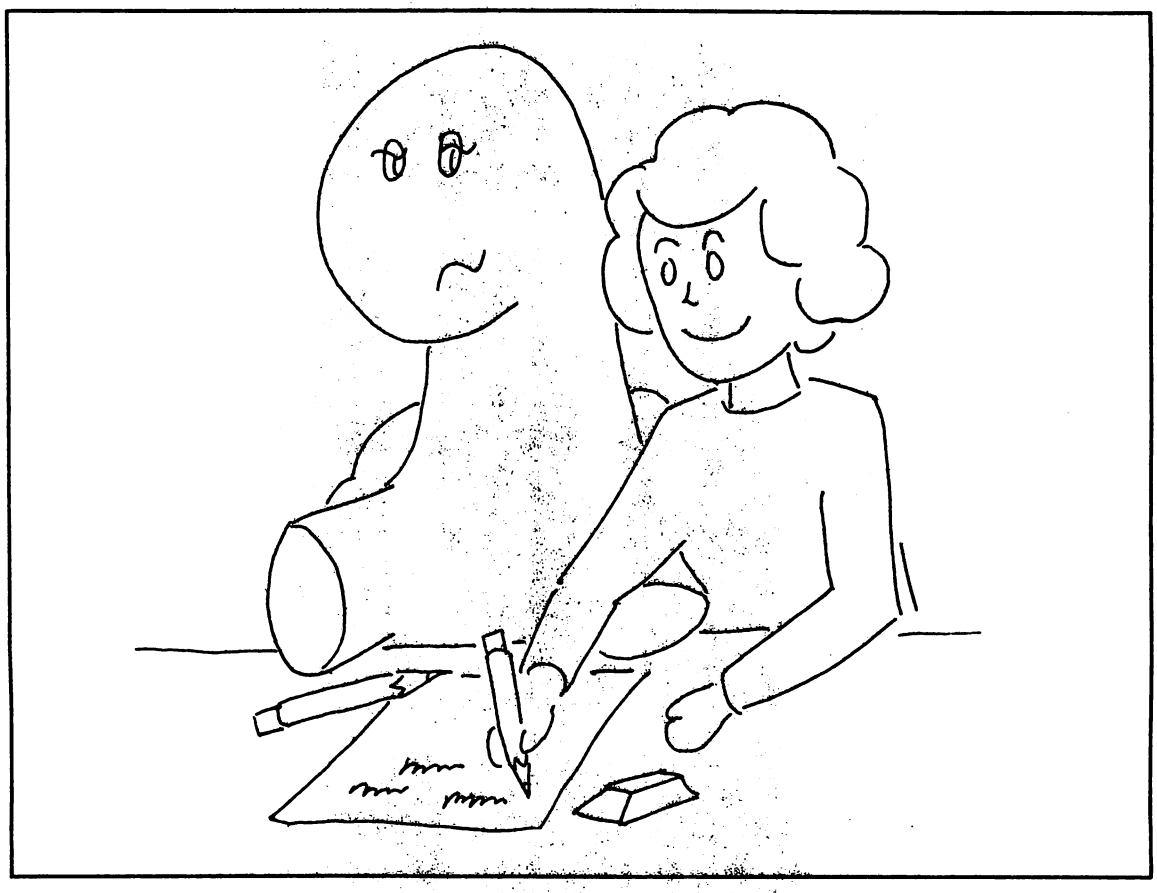
Another way of playing turtle is to do it with a friend or an adult. This way you can discuss what the possible solutions might be. Using these two powerful ideas, playing turtle and working as a team, is bound to be more effective than using one powerful idea. Using two ideas together is often more than twice as powerful as the two ideas used separately, just as having two people to help with a problem is more than twice as powerful as one person working on a problem.

WRITING OUT WHAT SAM SHOULD DO NEXT

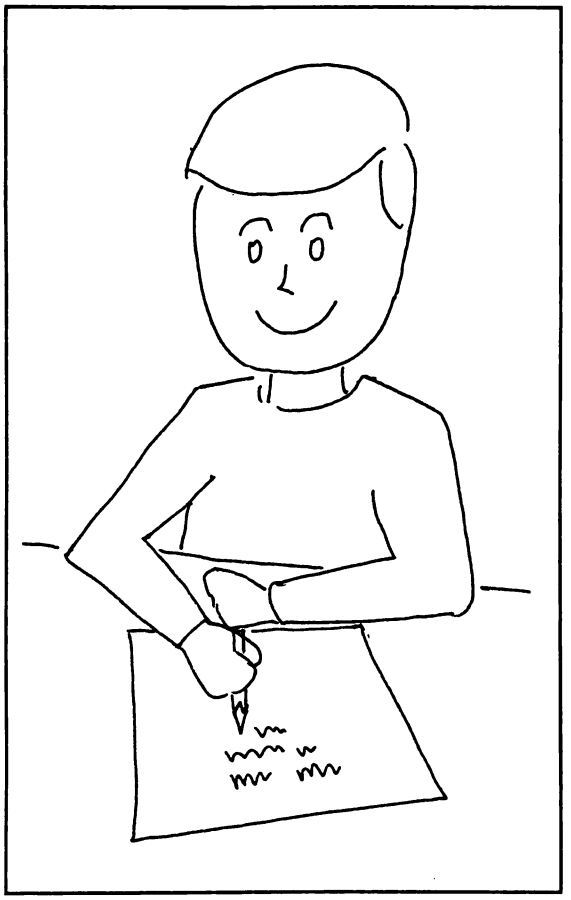
It is often helpful to write out what you want Sam to do in the form of a list. If for instance, you intended to draw a garden, your list might look like this.

Draw a tree
Draw a flower
Draw some grass
End

We will use this idea in the chapters that follow when we are talking about solving some of our problems. We will write out statements in English to help make our ideas seem simpler.



Helping Sam out with a problem!



Writing a list of English instructions for Sam.

RULES THAT SAM MUST FOLLOW

Perhaps you have noticed, that Sam seems to be fond of a particular group of numbers! In the procedures that we have used, some special things can be made to happen by using the correct combinations of these numbers. For example, if we want to draw a set of flags in a circle, we might have the following.

```
TO MANY.FLAG
REPEAT 6 [FLAG RT 60]
END
```

Or if we wanted to draw a closed triangle, we might use the following procedure.

```
TO TRI
REPEAT 3 [FD 100 RT 120]
END
```

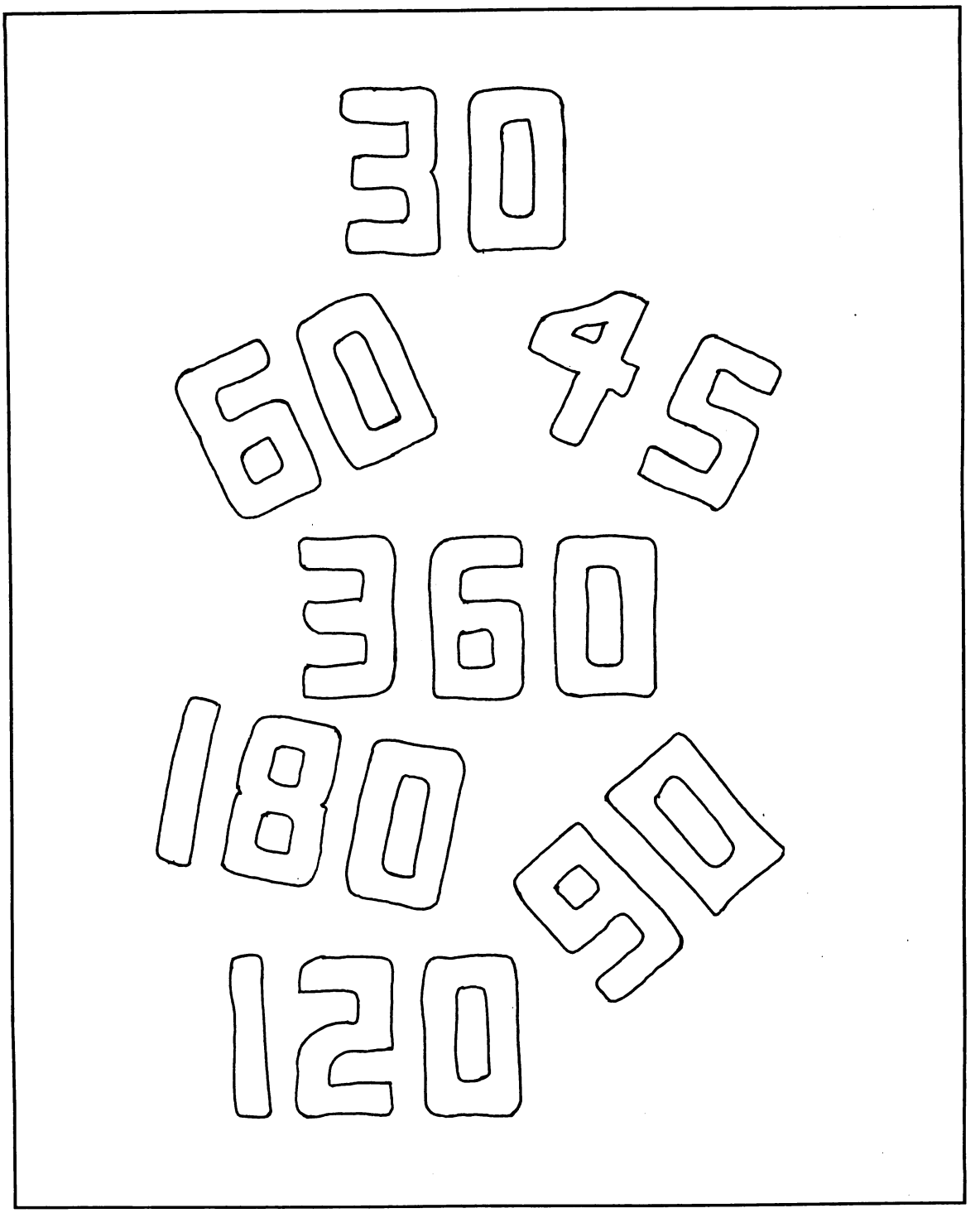
Can you see the special numbers? They are 6 and 60, and 3 and 120. What happens if you multiply 6 and 60, or 3 and 120? Correct, you get 360! This is one of Sam's special numbers! If you know a little bit about geometry, then you will know that 360 is also a special number in geometry!

If Sam makes a complete round trip, as he did when he drew the triangle or the flags in a circle, then he will have turned through exactly 360 degrees. If he makes TWO complete round trips, then he will have turned through two times 360 or exactly 720 degrees. What if he only makes one half a round trip? For one half a round trip, he would have turned through 180 degrees!

This means that the following number pairs, can give you interesting patterns in Logo.

2 and 30
3 and 30
12 and 30
2 and 45
3 and 60
4 and 90
2 and 90
3 and 120
2 and 180
1 and 360
2 and 360
3 and 360

There are of course many more combinations of numbers that will give you interesting patterns. Can you find some of them? Keep these special numbers in mind as you use Logo! If you are stuck, and you have run out of ideas, then you perhaps can ask the question



Some special numbers for Sam!

"How many degrees has Sam turned so far, can I use one of the special numbers to help Sam out?"

USING A TRACE FUNCTION IN LOGO

If you are having problems with a procedure, here is a practical utility for you to use when you are debugging that procedure.

One of the most powerful debugging tools in a compuer language is the *trace function*. Usually, you switch on the trace function by typing a single word. Sometimes the word might be TRACE, or it might sometimes be the word TRON. Hmm. . . I wonder where I have seen that word before?

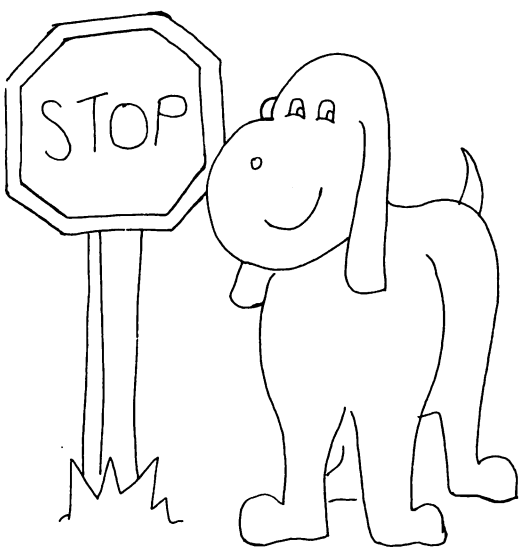
When the trace is on and you run a procedure, the computer will only do one step of the procedure at a time. This means that you can watch what is happening as your procedure is executed. Some computers will wait for you to press a key before it will continue to the next step! Other computers will execute all of the steps at one time, but they will give you a list of all the things that have been done, as they are being done.

When you use a trace function, you can examine a procedure very closely and if there is a bug in it, you can often figure out exactly where the bug is. After that, you can perhaps figure out what the bug is.

Unfortunately, Apple Logo does not have a built in trace function, but never fear, if you are willing to spend some time, then you can teach your computer how to trace a program by typing in a procedure from your Apple Logo manual. If you look at page 133 in the manual, you will find a procedure called STEP. Type this page of primitives into the computer very carefully. When you have done this, check the procedure line by line for errors. When it is checked, save it on disk by typing SAVE "STEP.

To use the STEP trace procedure, load the procedure that you want to trace into Sam's memory. Let's say that you want to debug the procedure BOX. You would load the procedure by typing LOAD "BOX. When the procedure loads, you will have your STEP procedures and your BOX procedures in memory at the same time. You then type STEP "BOX to set off the trace function. When the prompt and cursor reappears, type BOX. The procedure will now run one step at a time. It will run the next step only when you press the return key.

When you have finished with the STEP procedure, type UNSTEP "BOX to disable the trace function. You must use the UNSTEP command to restore your procedure BOX to it's original state.

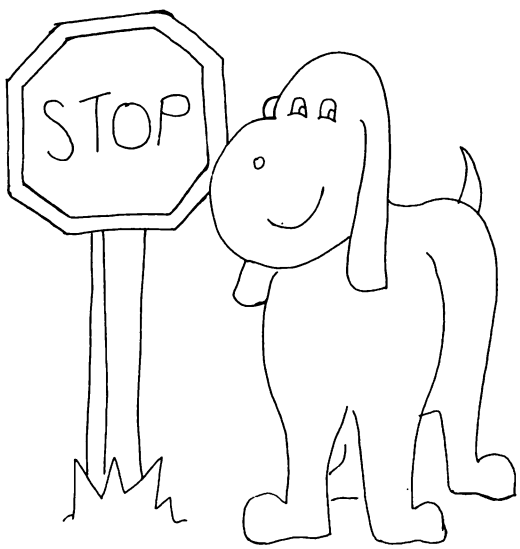


The STEP procedure will change the procedure BOX! After you have typed STEP "BOX, try looking at the primitives in BOX by typing PO "BOX. They will be quite different from the original BOX procedure. When you type UNSTEP "BOX, then the BOX procedure will be changed back into its original state. This is an interesting example of one computer program changing another program!

SUMMARY

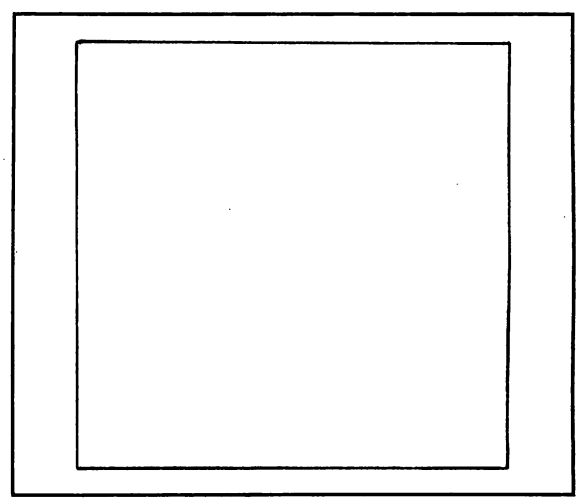
It is very useful in problem solving to have read a lot of articles in magazines and books. In Turtle geometry, looking at a lot of procedures that have been published in magazines and books can be very helpful. You should type in procedures from magazines that seem interesting, watch what they do, and look at them to see if you can discover how they work.

Finally, as we said before, there are very few right ways and wrong ways of playing at being turtle or using Logo. If an idea interests you, then go ahead and play with that idea. Logo is intended to be a computer language in which your ideas are the most important thing. Experiment with Sam, and then decide what you would like to tell him to do.



The idea of not getting it right the first time is well known in problem solving. Very few problems in real life are ever solved the first time. In fact, an error will often give you a further idea that you might find more interesting than the problem you are working on. By making mistakes and fixing errors, you will learn a lot with Sam.

1. Logo is intended to be fun.
2. When you use Logo, you will find that there are problems that arise that you will want to solve.
3. Problems that confuse Sam are often syntax errors.
4. Syntax errors are often caused by spelling mistakes or missing spaces. Some common syntax errors are listed below.
   - Leaving out spaces that are required in primitives.
   - Leaving out quotation marks.
   - Using the wrong brackets in statements such as REPEAT. The brackets required for REPEAT are generated by using the SHIFT-M and the SHIFT-N keys.
   - Leaving out numbers when they are needed. For example, FD must be followed by a number.
   - Misspelling primitive words.
5. Problems that confuse you are the how-do-you-do-that type of problems.
6. Problems that confuse you can be solved by:
   - Breaking the problem down into smaller parts.



Box.

- Looking for a similar problem that you know the solution for and using those ideas to solve your new problem.
- Using trial and error.
- Working from the answer backwards.
- Working with a friend towards a solution.
- Playing at being Sam.
- Writing out your ideas on a piece of paper in the form of a list.
- Using Sam's special numbers of 360 etc.
- Using the trace function from the Logo manual.

CHALLENGES

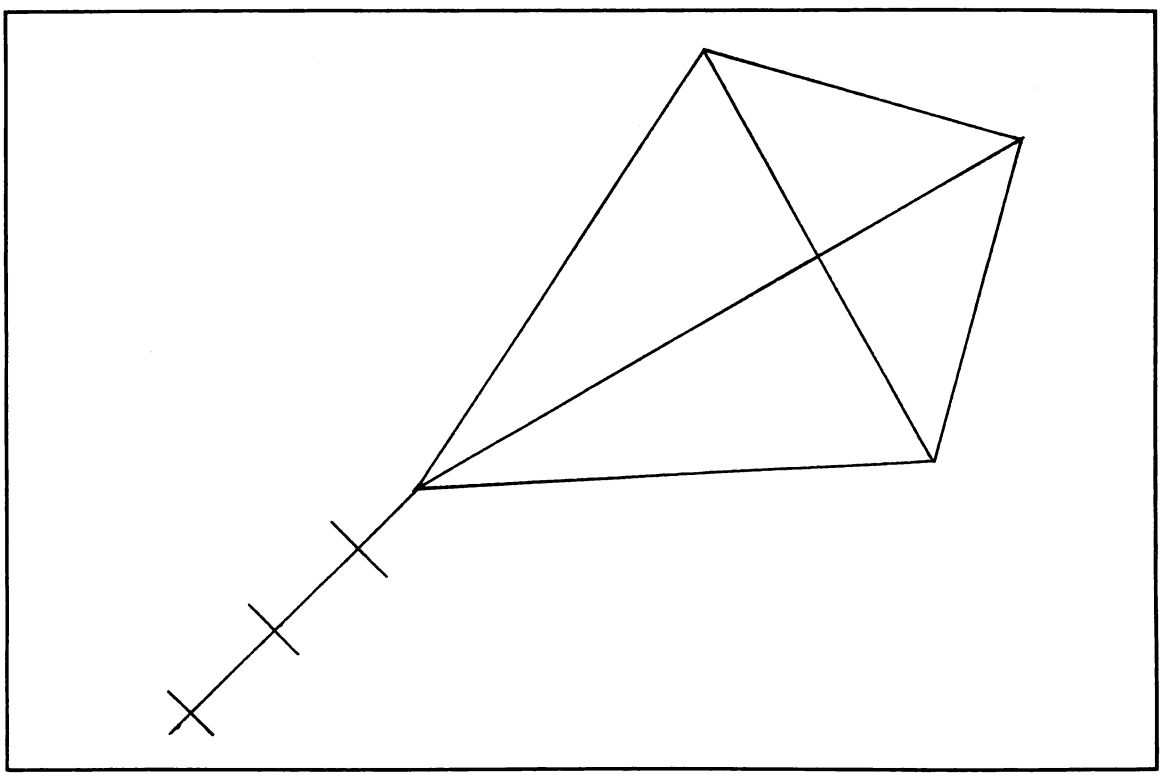
1. There is something wrong with the following procedure, can you figure out what it is and then make some changes so that it will run properly?

```
TOBOX
REPEAT 4 [FD20 RT 90]
END
```

2. There is something wrong with the following procedure. Can you figure out what it is and then make some changes so that it will run properly? The procedure was intended to draw the diagram on page 58.

```
TO BOX
RPEAT 5 [FD 100 RT 60]
END
```

3. There is something wrong with the following procedure. Can you figure out what it is and then make some changes so that it will run properly? The procedure was intended to draw the following diagram.



Kite.

```
TO KITE
TAIL
TRI2
TRI1
END

TO TRI1
REPEAT 4 [FD 30 RT 100]
END

TO TRI2
REPEAT 3 [FD 40 RT 100]
END

TO TAIL
RT 50
FD 30
END
```

4. Write an English list of instructions to tell Sam how to draw a tree.
5. Write an English list of instructions to tell Sam how to draw a motor car.
6. Write an English list of instructions to tell Sam how to draw a teddy bear.

QUIZ

1. Problems that confuse Sam are often _____ errors.
2. Syntax errors are often caused by _____ mistakes or missing spaces. Some common errors are.
   - Missing out _____ that are required.
   - Missing out _____ marks.
   - Using the wrong _____ in statements such as REPEAT. The brackets required are generated by using the _____ and the _____ keys.
   - Missing out _____ when they are needed. For example, FD *must* be followed by a number.
   - Misspelling _____ words.
3. Problems that confuse _____ are the "how-do-you-do-that" type of problems.
4. Problems that confuse you can be solved by:
   - Breaking the problem down into _____ parts.
   - Looking for a _____ problem that you know the solution to and transferring those ideas to your problem.
   - Using trial and _____.
   - Working from the _____ backwards.
   - Working with a _____ towards a solution.
   - _____ at being Sam.
   - Writing out your ideas on a piece of paper in the form of a _____.
   - Using Sam's special numbers of _____ etc.
   - Using the _____ from the Logo manual.

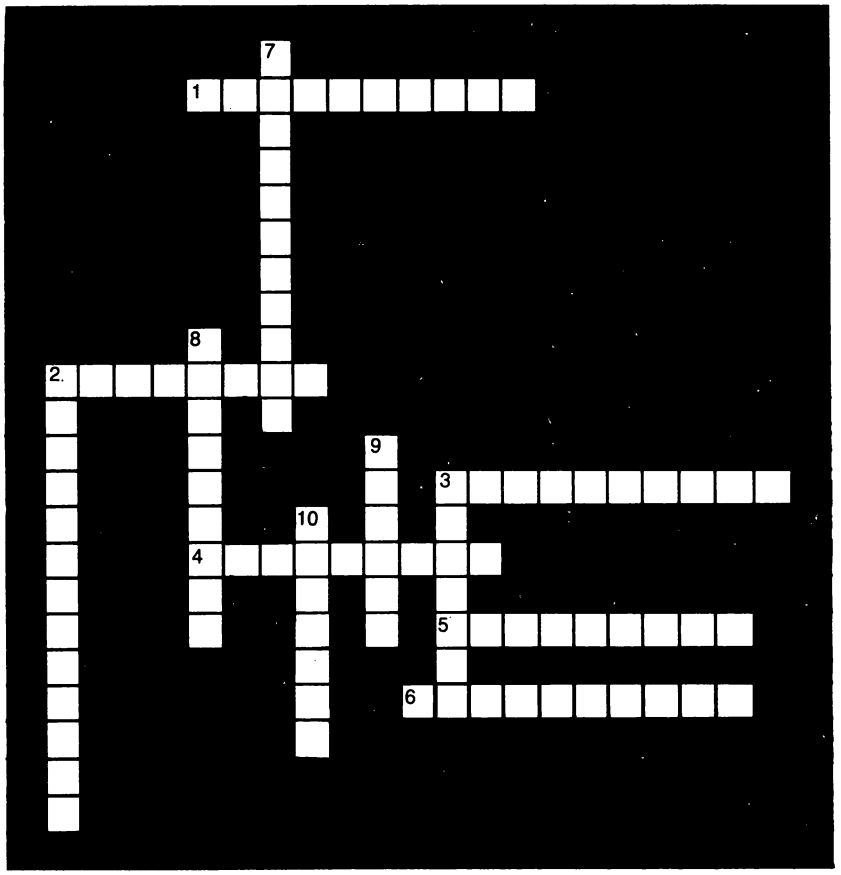
CROSSWORD PUZZLE

Clues Across

1. When you are trying to solve a problem, it is often a good idea to _____ of English statements.
2. When you see the prompt and cursor, you know that you are at this level in Logo.
3. When we pretend that we are Sam, we say that we _____.
4. This is something that computers and machines don't have.
5. When Sam does what we want him to do, we know that _____.
6. If a problem is long or difficult, break it into _____.

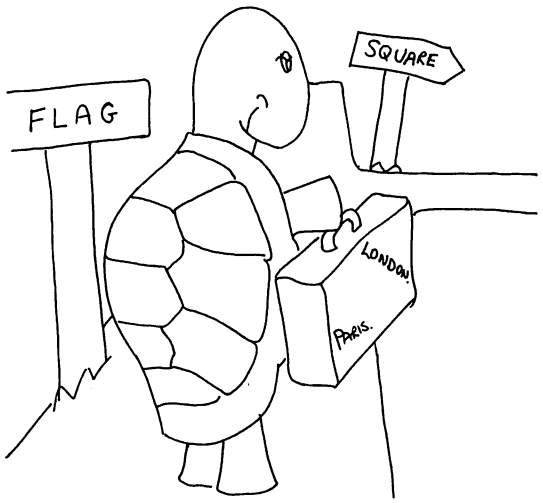
Clues Down

7. You can sometimes solve problems by looking at these!



8. This is a word that means we are fixing a procedure.
2. This method of solving problems can take a long time!
9. If we miss out a space, or misspell a word, we call this a _____ error.
3. This is what we call something difficult.
10. This is what a procedure has when it won't work.

Chapter 4



If You Don't Type Well, You Will Love This!

Many of us don't type well. Even those of us who do type well often want to make changes to the programs or procedures that we are working on. In the previous chapters, we have been telling our turtle, Sam, to do a variety of things. If we made a mistake, we simply typed the procedure or primitive over again. Also, if we decided that we wanted to add something more to a procedure, we had to type the whole procedure once more, even if the new one was a lot like the old one. Logo provides us with a way of changing what we have already typed in as a procedure. Logo uses an *editor* to allow us to make these changes.

The editor is complicated to use to begin with, but once you know how to use it, you will find that it is a tool that you will use all of the time. Editors are so powerful that almost every language system that a computer uses is supplied with one, so that the programmer can change the typed lines in his program easily.

CONTROLLING SAM'S MEMORY AND SCREEN

Before we continue, we should talk about some commands that you will need to control Sam's memory and screen.

Erasing Procedures from Memory

The first commands that we should look at are, ERPS and ER. ERPS is short for ERase ProcedureS and ER is short for ERase. If you type.

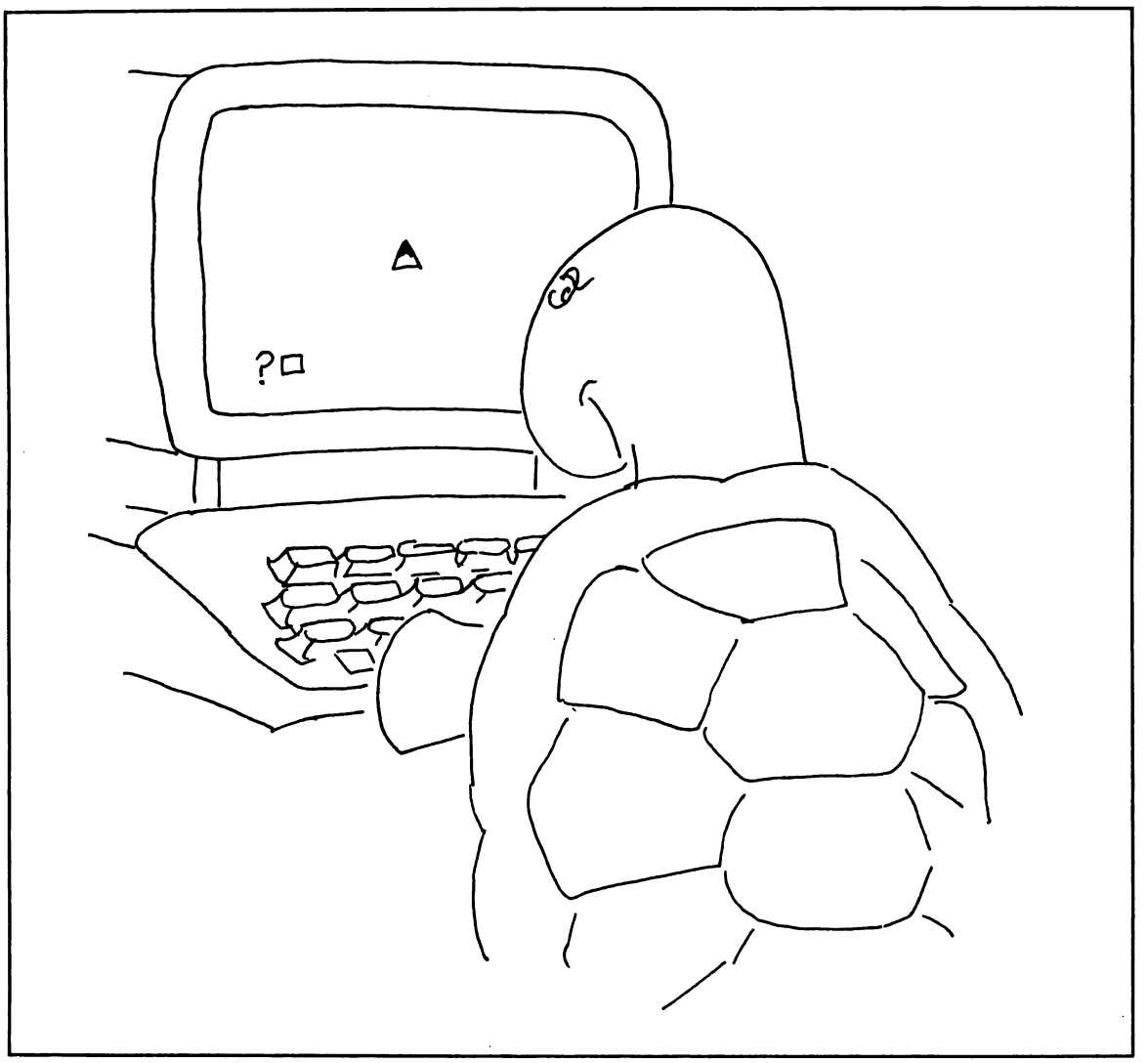
```
ERPS
```

then *all* of the procedures in Sam's memory will disappear.

Likewise if you type

```
ER "BOX
```

you will find that the procedure BOX will be



Sam likes to type at the keyboard.

erased! However, *none* of the other procedures will disappear. You should use the ER primitive if you want to erase only some of Sam's memory.

Finding Out What Is In Sam's Memory

Here are some primitives that you will find very helpful when you are using procedures. They will allow you to look at the names of the procedures that Sam already knows. You will then be able to look at the primitives that make up these procedures.

If you want to see the procedures that are still in Sam's memory, you can type

POTS

This will give you a listing of the things that Sam has in his memory. If you recall, you can put things in Sam's memory either by typing

procedures at the keyboard or by loading procedures from the disk.

If you want to look at the primitives inside the procedures, you type the following

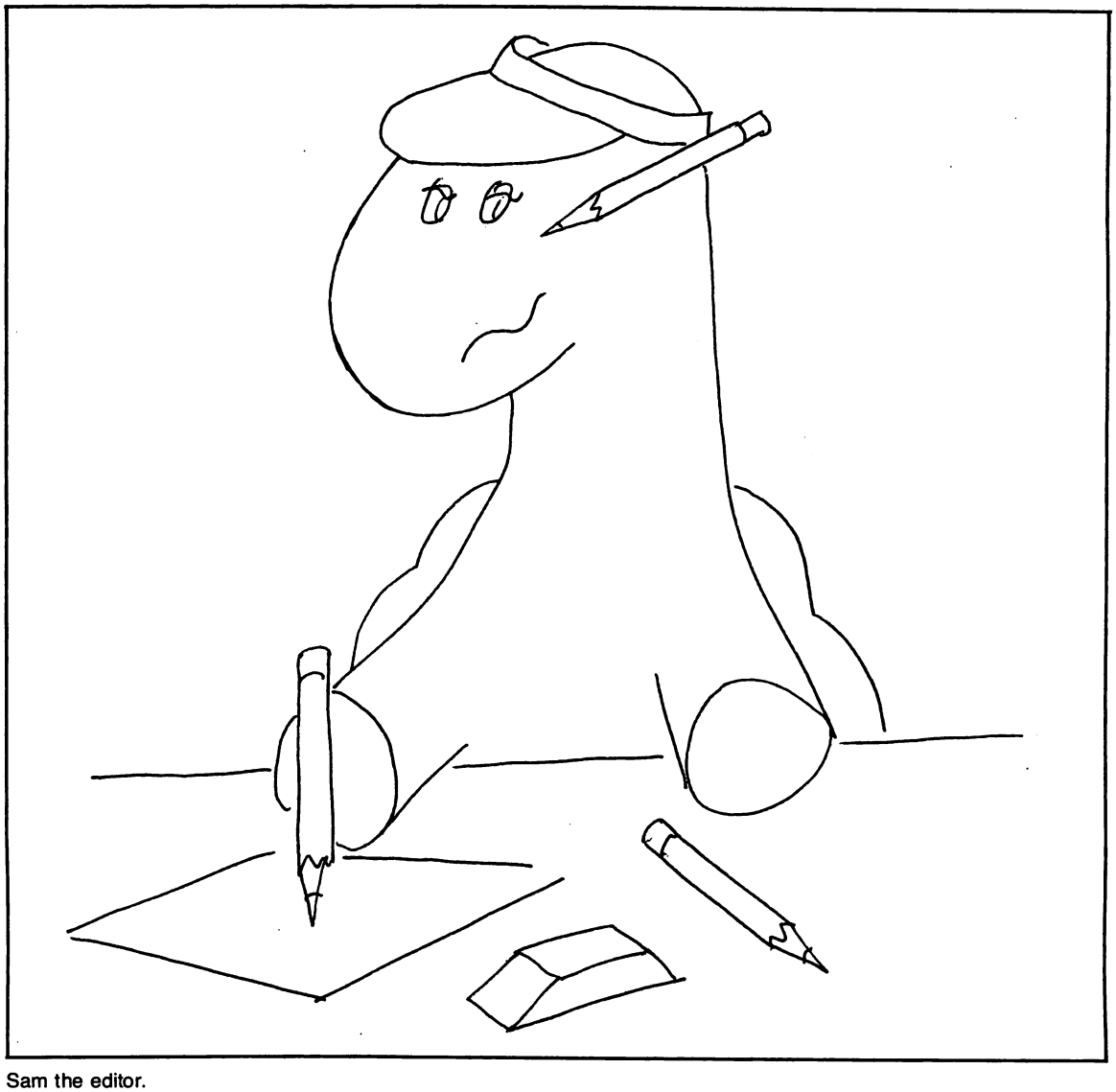
POPS

The primitives that make up each procedure will be listed on the screen when you press the return key. Unfortunately, the words "disappears" behind Sam's drawing area.

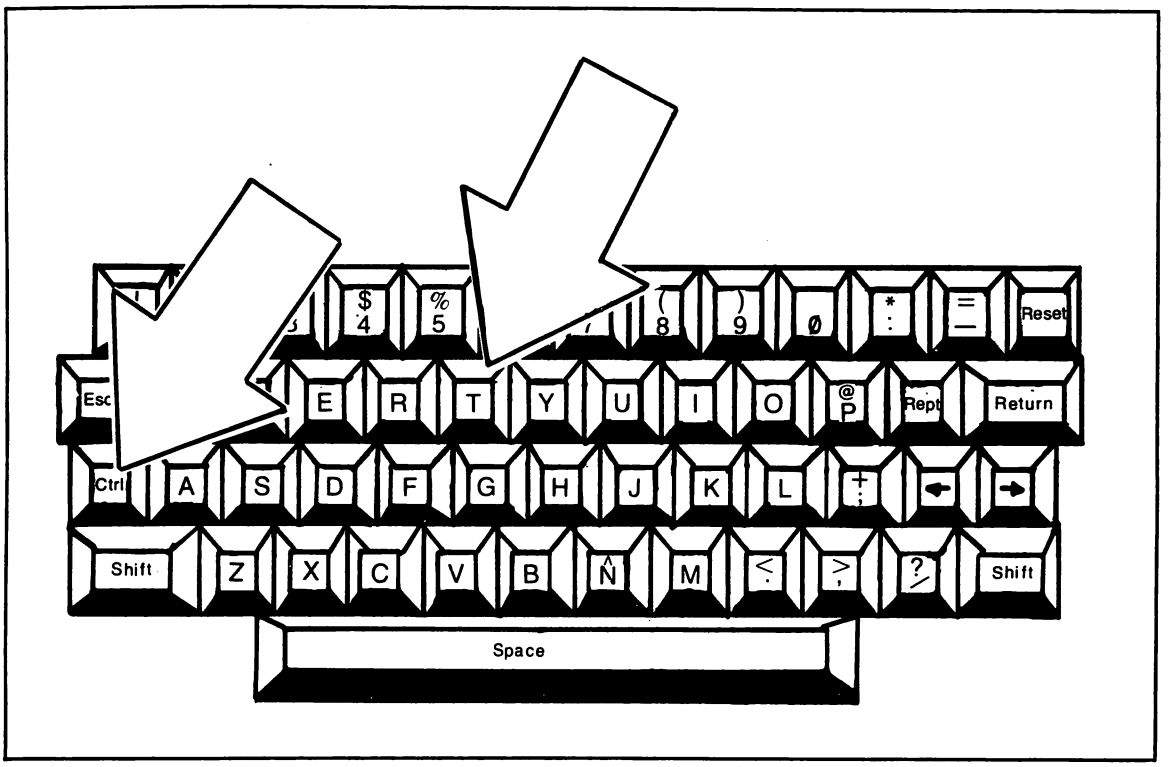
Changing the Display Screen

You can make the words, or *text*, reappear by typing

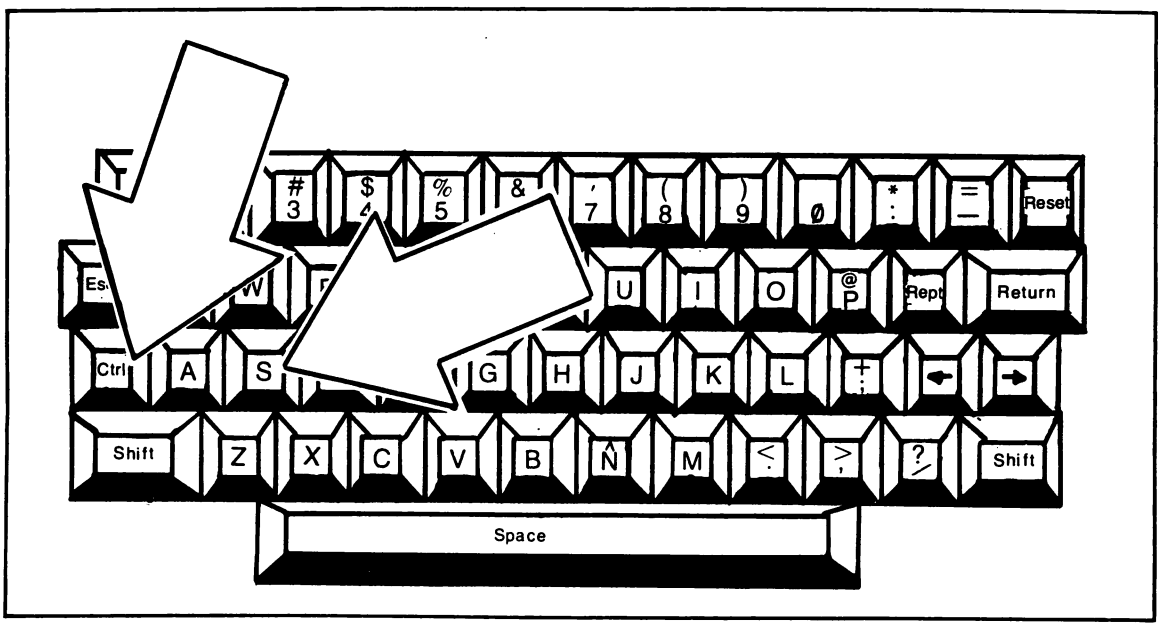
CTRL-T



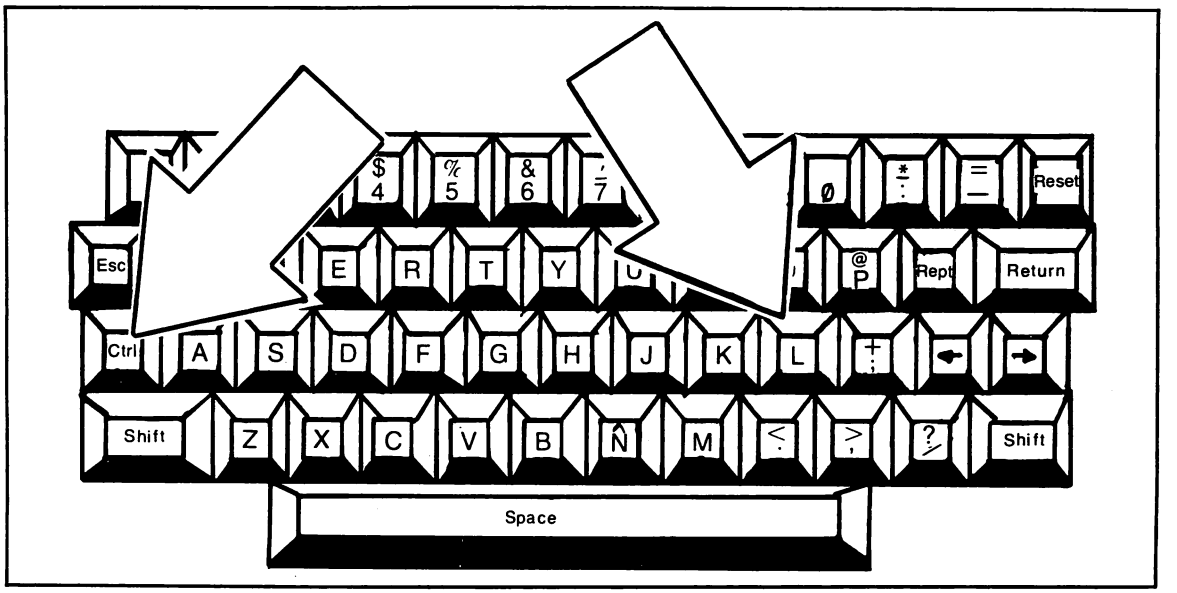
Sam the editor.



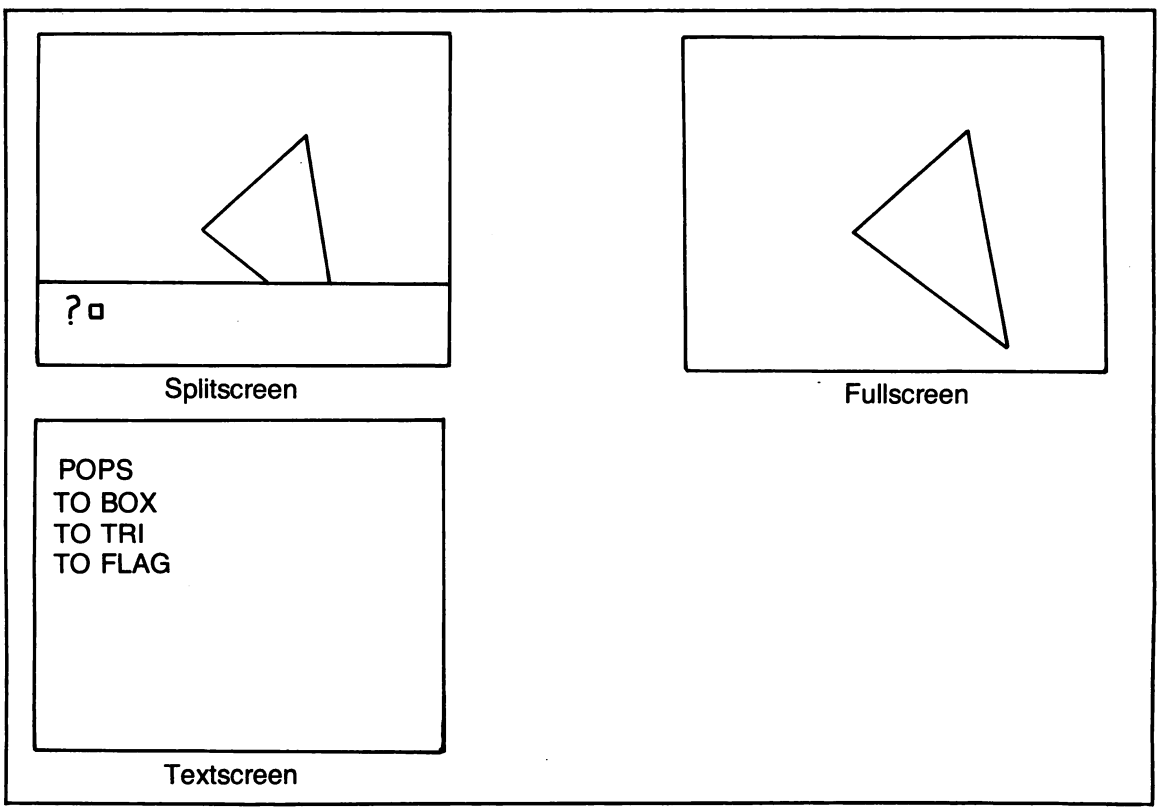
This will give you textscreen.



This will give you splitscreen.



This will give you fullscreen.



Here are the different screen modes.

The CTRL key is on the left-hand side of the keyboard. To use CTRL-T, hold down the CTRL key, and then press the T key. This will give you the **textscreen**, when you go into the textscreen mode, the *graphics* screen that you have been drawing on will disappear. If you want to go back to the graphics screen, but still have four lines of text at the bottom (called the SPLITSCREEN), you should type

CTRL-S

While we are playing with Sam's workspace, try typing

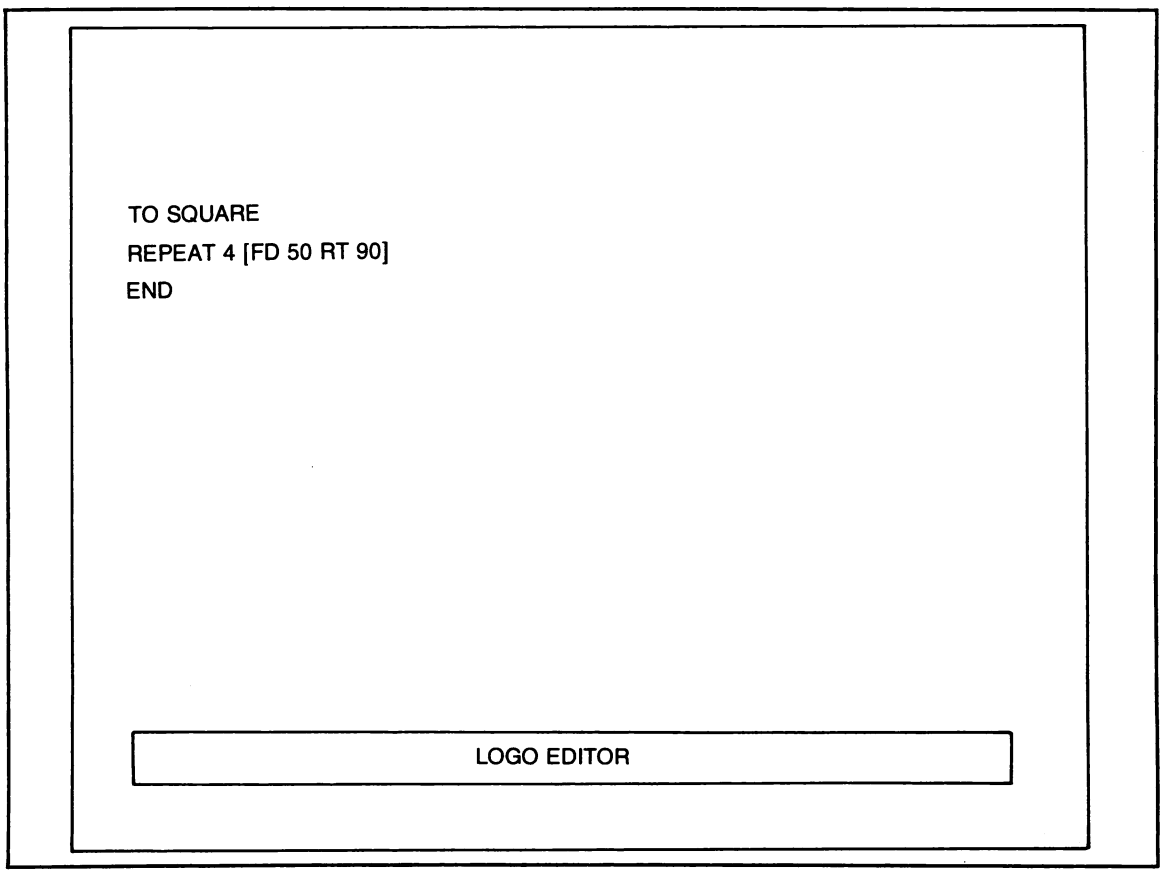
CTRL-L

This will give you the *fullscreen* (You can think of the L as standing for Large if you like). The fullscreen will show only Sam and his graphics, with no text at all. It is nice to have only Sam on the screen sometimes, you can plan on bigger designs that way! Of course, if you want to return to the "regular" screen, type CTRL-S again.

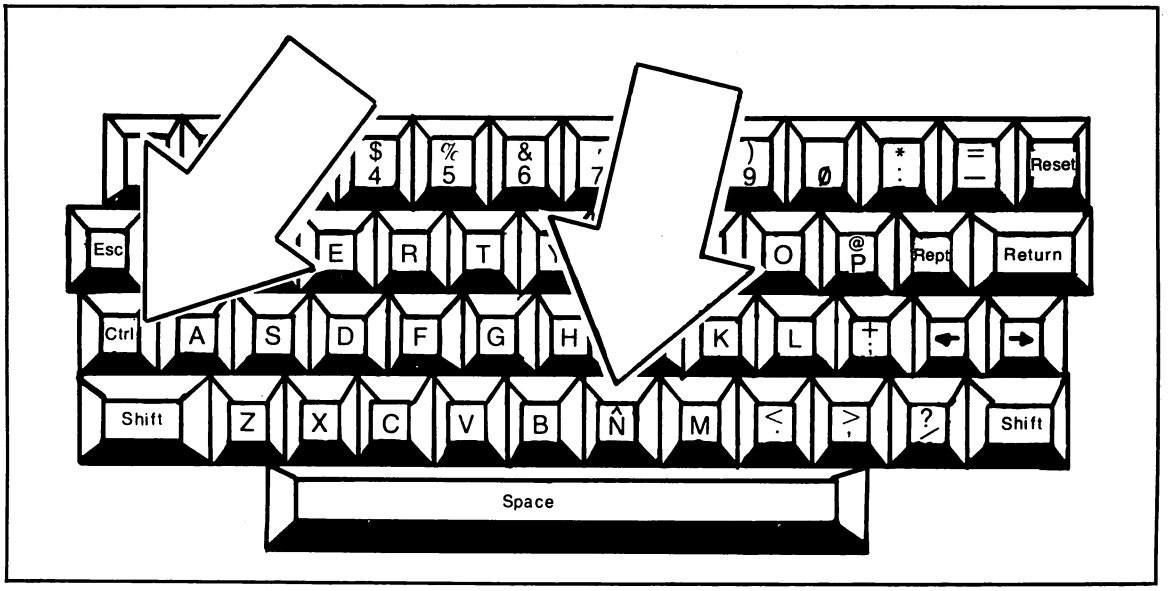
Now that we have looked at POTS, POPS and at some screen control keys, lets look at some specific ways to use the editor.

USING EDIT "SQUARE

To illustrate how the editor works, we will load the procedure SQUARE from disk. If you do not have the procedure SQUARE on



This is what the edit screen looks like.



This will move the edit cursor to the next line in your procedure.

disk, look in Chapter 2 and type in the procedure. When it is typed in, save it on disk as described in the chapter.

To load the procedure from disk, type.

```
LOAD "SQUARE
```

To enter the editor, type the following:

```
ED "SQUARE
```

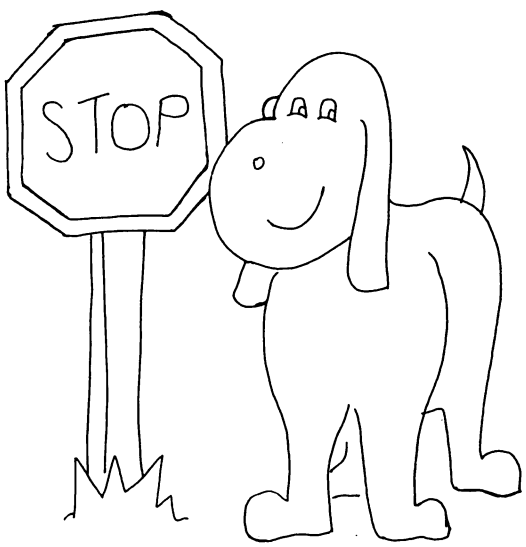
When you press the return key, the screen will change. It should look like the screen on page 67.

Sam has disappeared for the moment. You cannot tell Sam what to do directly from the editor. You have to leave the editor in order to get back to where Sam lives.

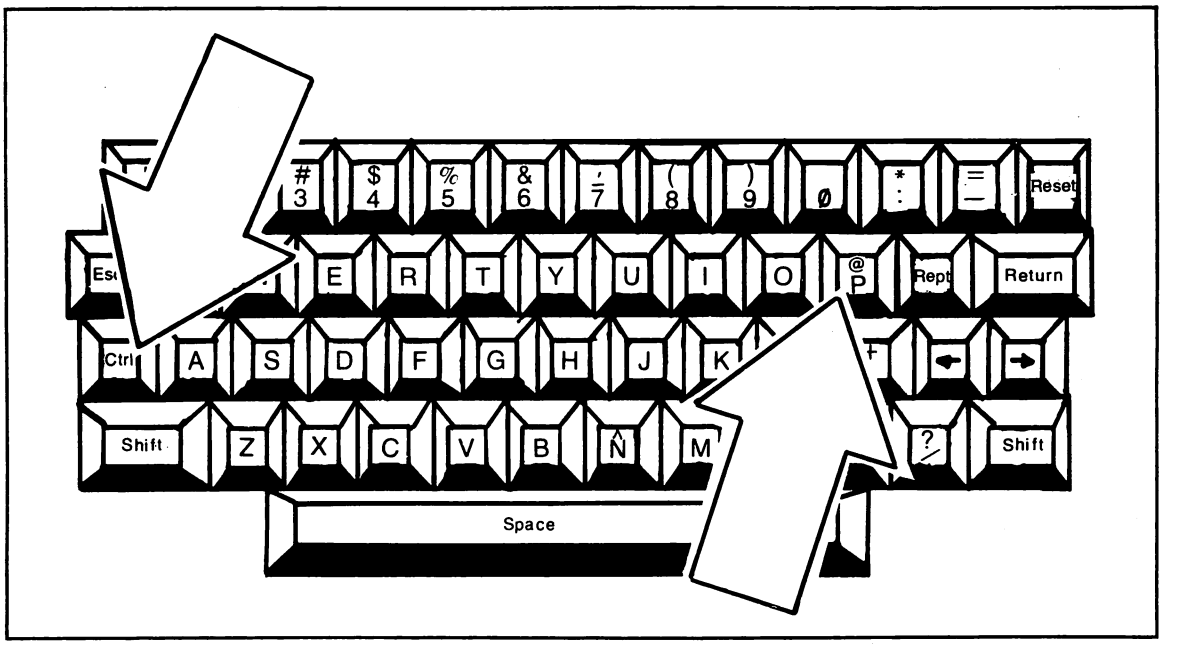
You will see a cursor on the screen. The editor is "looking" at your procedure at this position. Start by moving the cursor around in your procedure.

Moving Up and Down in Your Procedures

Press the CTRL key and then press the N key. You should see that the cursor moves down to the next line of the procedure. Do this again to satisfy yourself that CTRL-N will always do the same thing. The letter N in CTRL-N stands for Next.



We used *shift* N to make opening brackets when we told Sam to REPEAT. However



This will move the edit cursor to the previous line in your procedure.

CTRL N will move the cursor to the next line when we are using the editor.

How can you move back up through the lines of the procedure? You can move back up through the lines by pressing CTRL key and then pressing the P key. The cursor will move back up one line. Just as the N stood for "Next line," the P stands for the Previous line. Practice using the CTRL-N and CTRL-P key, so that you are familiar with these cursor motions. Now that you can move up and down the procedure, how do you move forward and back on the primitive lines in the procedure?

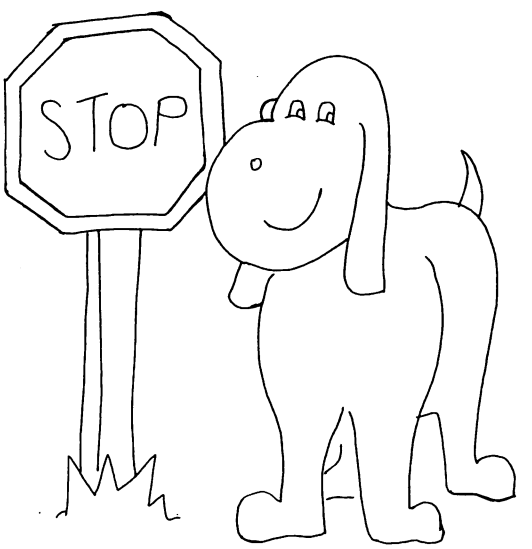
Moving Forward and Back in Your Procedure

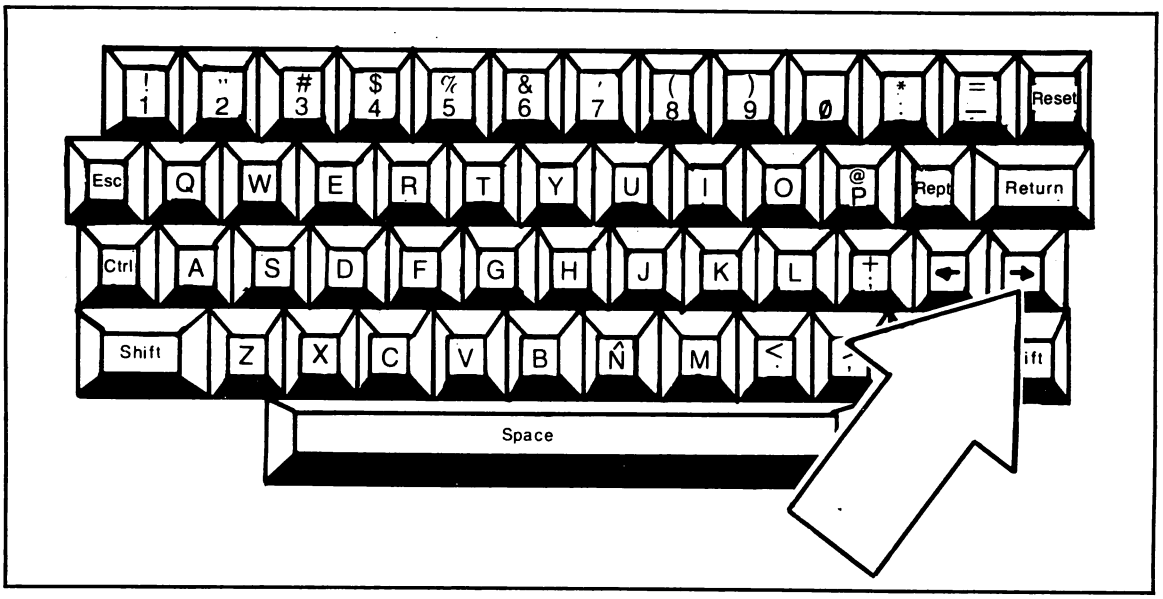
WATCHOUT!!! Using the back arrow key *swallows* the text to the left.

Try pressing the right arrow key on the right-hand side of the keyboard. This should move the cursor to the right.

BE CAREFUL!!! the back-arrow key does not move the cursor to the left. If you want to move the cursor to the left you must hold down the CTRL key and press the B key. Guess what the B stands for. Correct, it stands for Back!

If you want to move *all* the way back to the



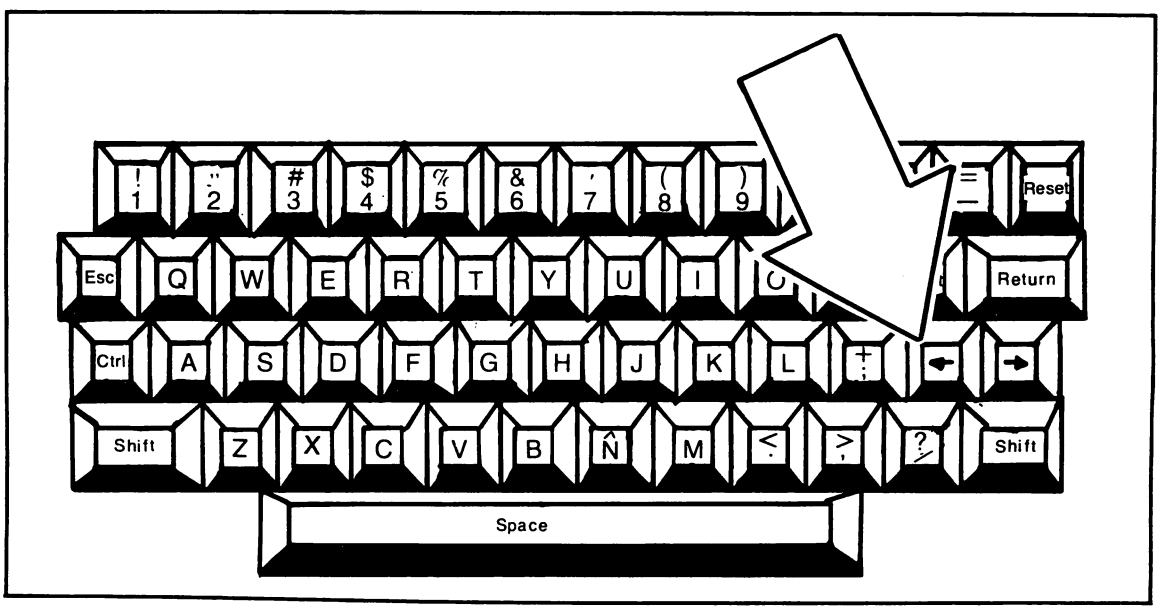


This will move the edit cursor forward.

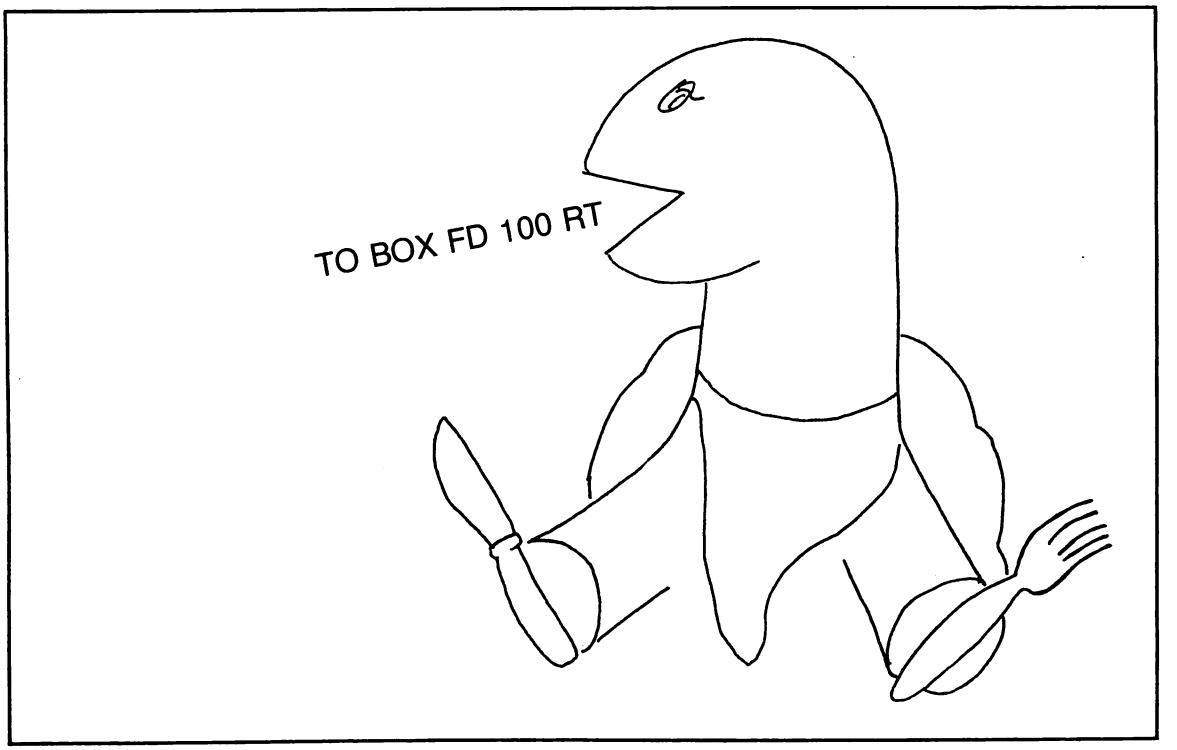
beginning of a line, you can use CTRL-A. A is the *beginning* letter in the alphabet. When you remember that, then the A in CTRL-A makes a lot of sense.

If you want to move to the end of a Logo line, use CTRL-E. Happily, E stands for End, so that is an easy one to remember!

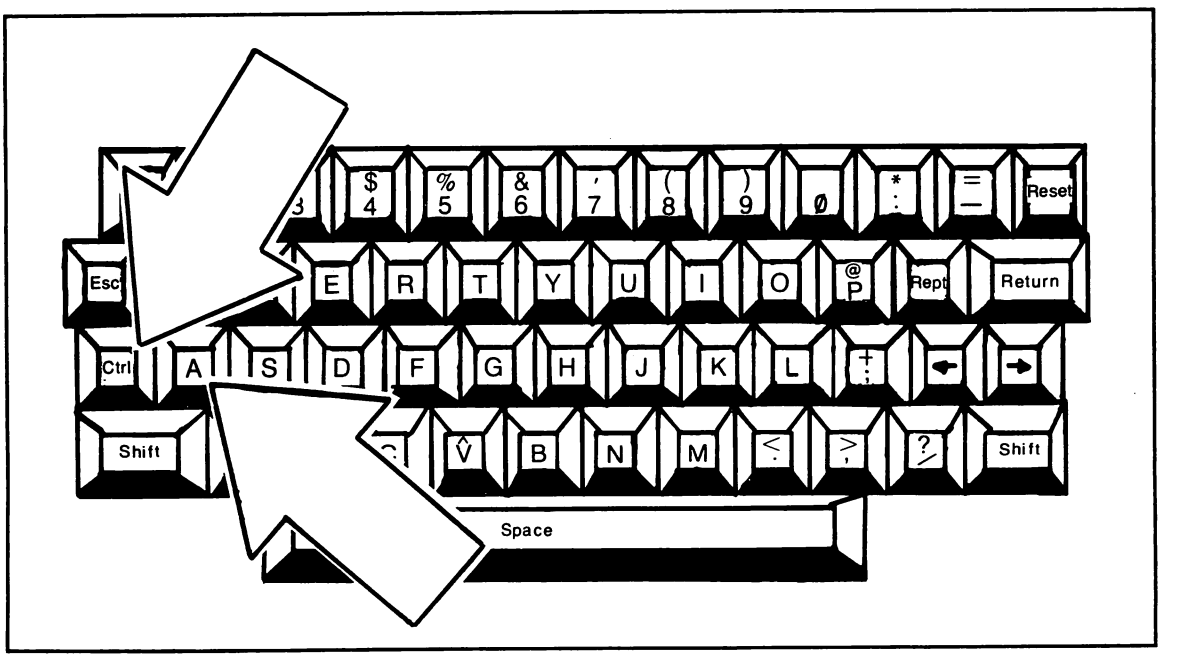
You can now move across and back on the lines of a procedure, lets look at how to rub characters out.



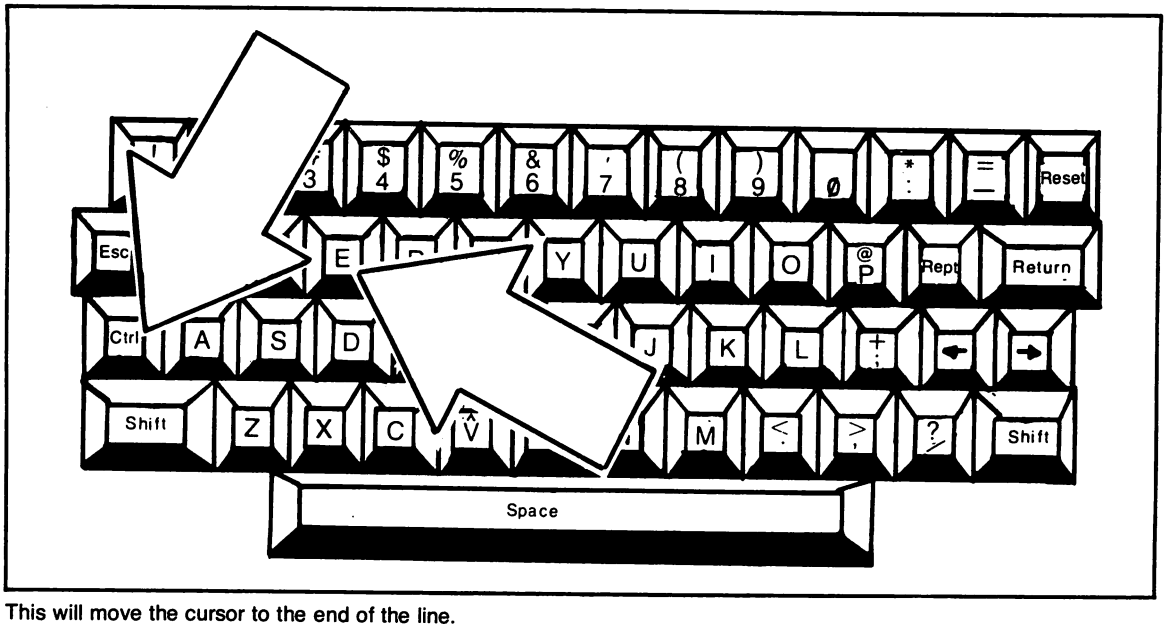
This will "swallow" your text!



Pak-Sam eats text if you use the back-arrow key!



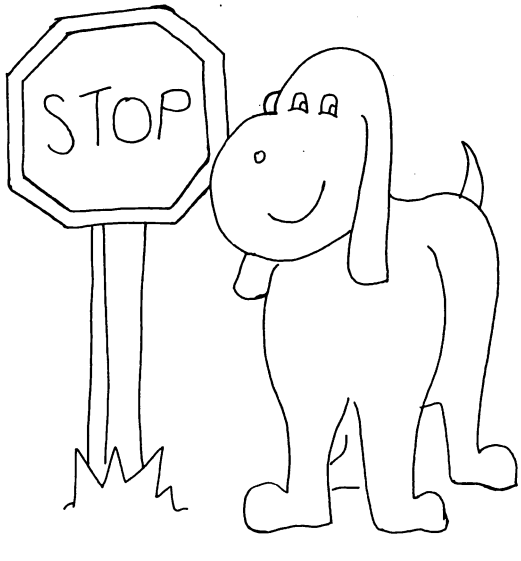
This will move the cursor to the beginning of the line.



This will move the cursor to the end of the line.

Rubbing Things Out

As we said, the back-arrow key swallows the text to the left of the cursor! Oops! If you have swallowed some text, simply move the cursor to the place where the text was swallowed and then type in the missing letters. You will find that the other text items are "pushed over" when you do this.



If you have swallowed a lot of text from the procedure, and you want to start over again, you can do so by tping CTRL-C and then reloading the procedure SQUARE from disk by typing LOAD "SQUARE. Here are the keystrokes that you would have to type to do this.

```
CTRL-C
ERPS
LOAD "SQUARE
ED "SQUARE
```

If you have a lot of text you want to "eat", hold down the back-arrow key and then press the REPT key. Please be careful not to eat the entire procedure!!!

One other way of rubbing out characters is to hold down the CTRL key and press the D key. The D of course stands for Delete, so that is another easy one to remember!

Pressing CTRL-D will delete only one character. If you want to delete more than one

character, use the CTRL-D and the REPT key together.

So much for erasing characters, but how do we insert things?

Inserting Lines That You Forgot About

If you have missed a few characters in the middle of a line, you simply move the cursor to the place where you want to begin inserting, and then begin typing the missing characters. The new characters will push the old ones over as you type them at the keyboard.

However, if you want to insert a new line, you simply move the cursor to the place where you want to begin the new line, and press the return key at this position.

You can also insert a line by typing CTRL-O. This Opens a new line. The O in CTRL-O stands for the word Open.
When you have succeeded in opening a new line, you can insert primitives by simply typing them in at the keyboard. When you reach the end of the line, press the return key.

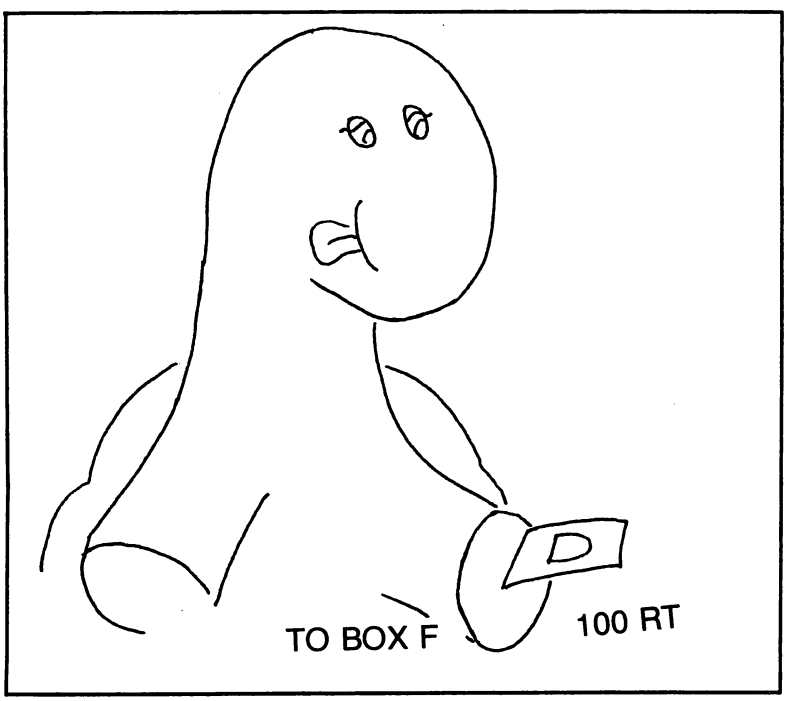
Getting Back to Where Sam Lives

Now there is something else that we have not looked at. What can it be? Of course. . . How do I get out of the edit mode so that I can tell Sam to do things once more? SIMPLY HOLD DOWN THE CTRL key and then press the C key. If you like, you can remember this by thinking that the C stands for Complete.

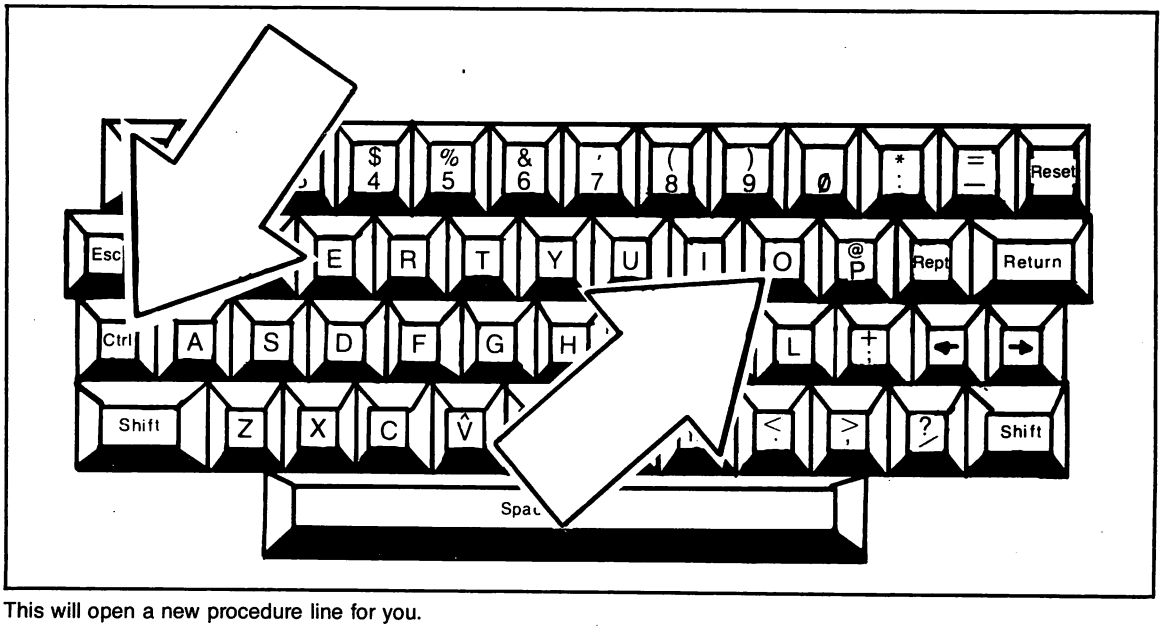
There are other keys that you can use for editing, and these are listed at the end of this chapter as well as at the end of the book in Appendix D. However, you can get along quite well with the edit commands that we have looked at so far.

TUTORIAL

Here is a short tutorial to show you how to



Sam can insert missing text for you!



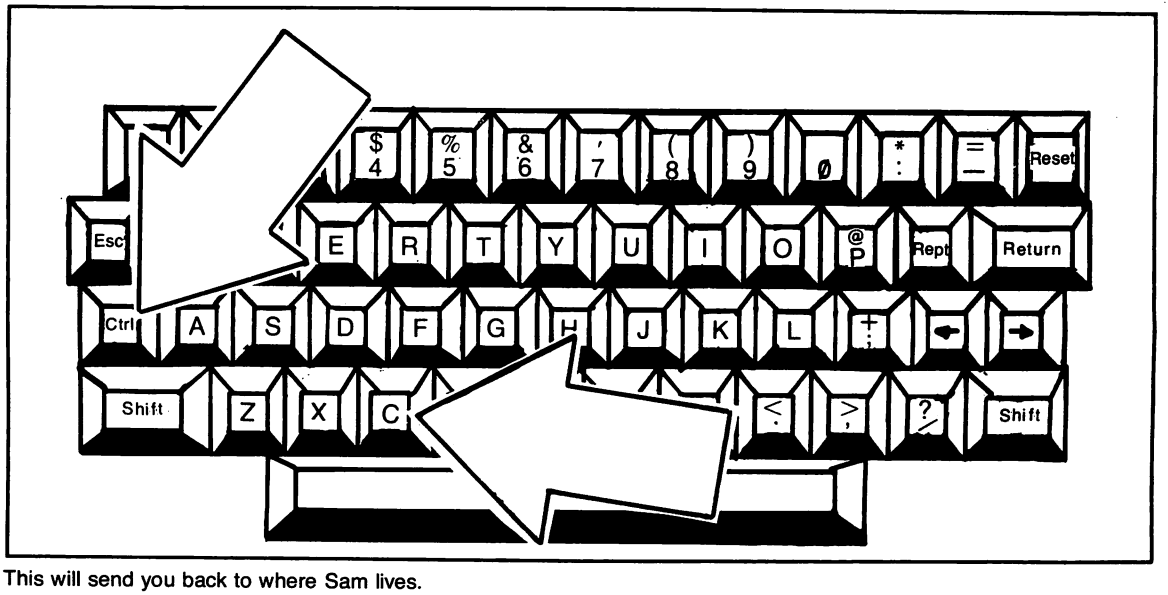
This will open a new procedure line for you.

use the editor. So that we all start from the same point, use CTRL-C to exit the editor; type ERPS, and then type LOAD "SQUARE. When the procedure has loaded, type ED "SQUARE to enter the editor.

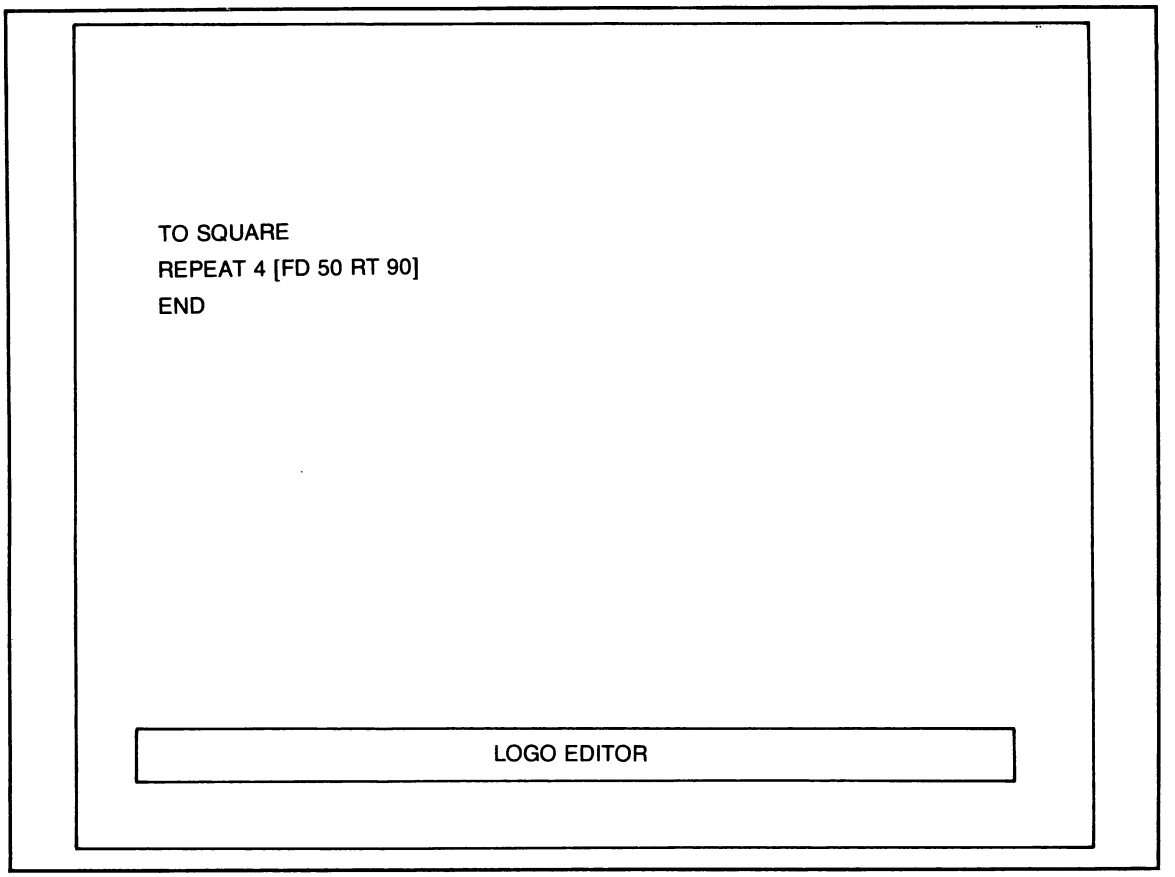
When you are in the editor, you will see the following procedure.

```
TO SQUARE
REPEAT 4 [FD 50 RT 90]
END
```

Suppose you want to change the number 4



This will send you back to where Sam lives.



Editing the procedure SQUARE.

in REPEAT 4 [FD RT 90] to the number 8. (The procedure would no longer give you a square if you did this of course! but try it anyway!) Move the cursor down to the second line by typing

CTRL-N (once)

Move the cursor to the right by typing

right arrow (seven times)

Replace the 4 with the 8 by typing 8 at the keyboard.

Oh oh . . . we now have an 84 in our procedure! The 8 was inserted where you wanted it, but the 4 didn't disappear! It was pushed along by the inserted character. Press CTRL-D to erase the 4. There, now you have exactly what you wanted!

Suppose you want to insert a new line, FD 20, just before the END statement. Type the following.

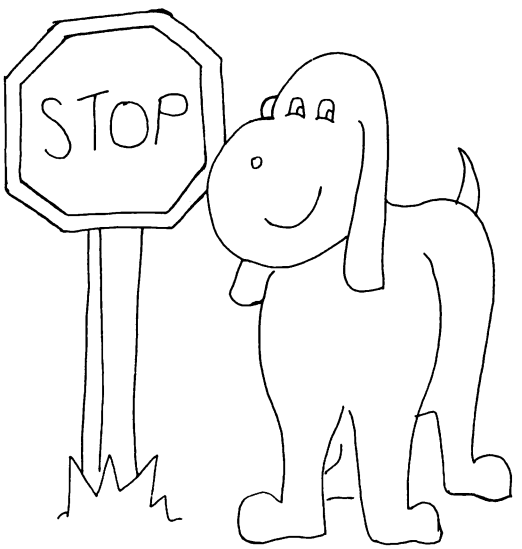
CTRL-N
CTRL-B (3 times)
CTRL-O (to open a line)
FD 20 (the new line)

If you now want to erase the FD 20, simply press the back-arrow key six times. This will swallow the FD 20.

That briefly is how to edit a procedure. As I said before, the few edit commands that you have looked at should be enough to get you by.

RENAMING A PROCEDURE

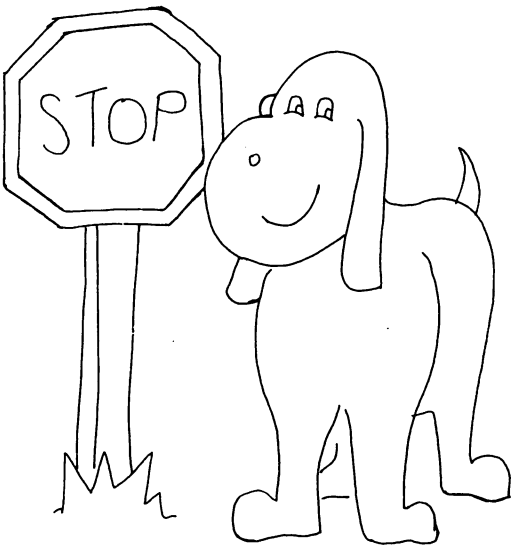
HERE IS HOW TO RENAME A PROCEDURE



Here is an important item. What if you wanted to change the name of a procedure? First of all, you should enter the editor in the usual way. Then you should move the cursor until it is on the name of the procedure, type in the new name of the procedure, and then swallow the old name. You can swallow the old name by using the CTRL-D key or by using the back arrow key.

Presto!!! When you type CTRL-C to exit the editor you will get the message that the procedure has been defined under its new name!

If you do rename a procedure, you should be aware that the old procedure will *still* exist in Sam's memory. To completely eliminate the old procedure and its name, use the ER primitive.



Try changing the name of the procedure SQUARE to BOXY to check that this will work.

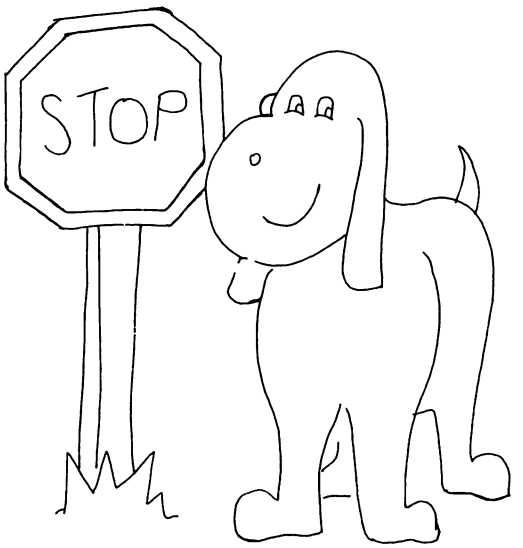
If you want to save the BOXY procedure for later, you will need to send it to a disk by typing

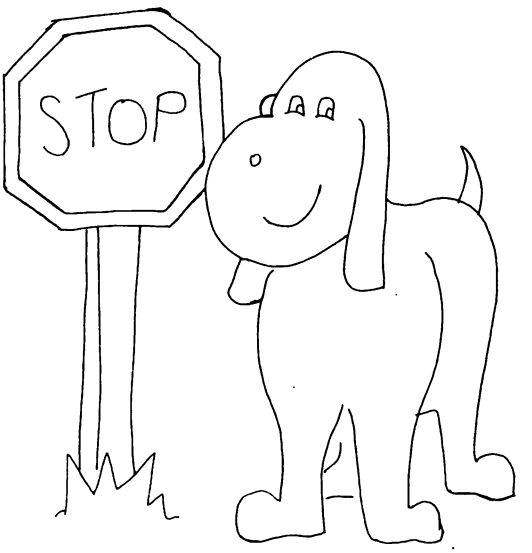
```
SAVE "BOXY
```

If you try to save BOXY by typing SAVE "SQUARE, you will be told by Logo that the file SQUARE already exists. Apple Logo will not save a file under the same name twice. If you want to use the same name for your file, then you can do this by typing ERASEFILE "SQUARE. This will rub out the old file called SQUARE and allow you to save your new file on disk under the same name.

DEFINING A PROCEDURE FROM THE EDITOR

There are three ways of teaching Sam about a procedure. Here is one that we have not looked at yet.





Here is how to define a *new* procedure using the editor
If you are in the editor, type CTRL-C to return to where Sam lives.

Lets say that you want to create a procedure to draw a pyramid. You should type

```
ED "PYRAMID
```

Logo will send you to the editor. At the top of the edit screen, you will see the words

```
TO PYRAMID
```

You may then type your procedure for PYRAMID as usual, remembering to close the procedure with an END statement. Then type CTRL-C to leave the editor as before.

Logo is very kind to us. It will supply us with an END statement if we happen to forget! There is no extra charge for this service!

Up to this point, we have looked at a number of ways of opening a procedure in Apple Logo. Creating a procedure is the same as teaching Sam about something of course. Can you remember the three ways that Sam can be taught things? Here they are

- By using the TO command.
- By using the editor and name of the procedure.
- By loading an existing procedure from disk.

SOME ADVANCED EDITING FEATURES

If you have found this chapter quite easy, then here are some more editing commands that you might find useful.

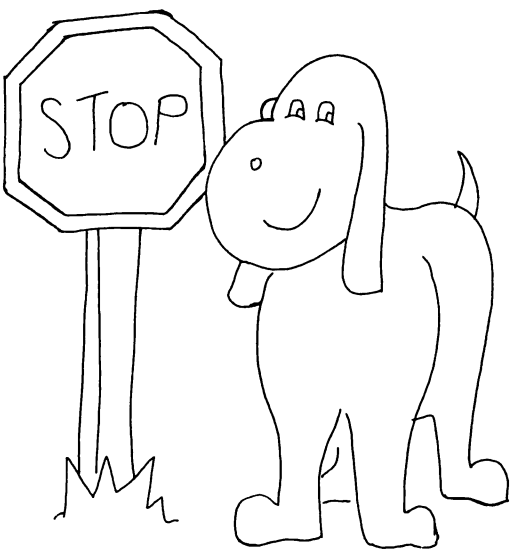
Killing and Yanking in the Editor

In Apple Logo, you can use two control keys together. They are the CTRL-K key and the CTRL-Y key. The K stands for Kill, and the Y stands for Yank. If you type CTRL-K from the editor, the text from the cursor to the end of that line will disappear.

The text has been placed in a *buffer*. A buffer is a part of the computer's memory that is used to store things temporarily. The text has not disappeared completely. To get the missing text back, type CTRL-Y. The text will

reappear, beginning at the cursor, as if by magic!

What does this really mean! It means that you can copy and move lines of text from one part of a procedure to another. If you are debugging a procedure and you find that you have a line of primitives in the wrong place, you can use CTRL-K and CTRL-Y to move the line to its correct position.

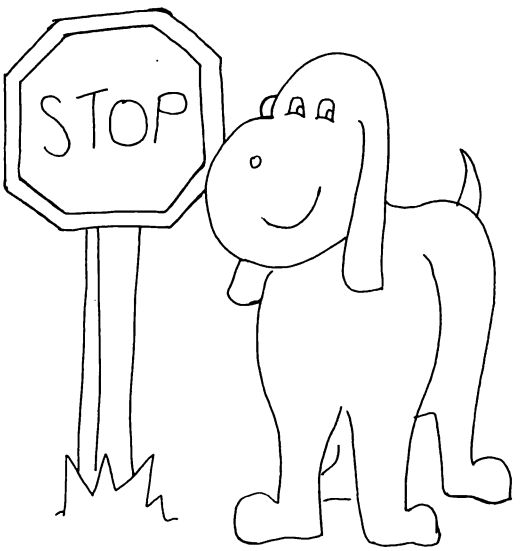


You can use some of the edit control keys when you are using Logo, just like you can in the editor. CTRL-Y and CTRL-K are particularly useful and will save you a lot of typing when you are teaching and telling Sam to do things.

If you have been editing a procedure and you decide that you do not want to go through with the changes that you made after all, you can tell the editor to forget all of the changes that you have typed by using CTRL-G. CTRL-G will return you to Logo with your procedure unchanged. What if you type CTRL-G by mistake? Easy, simply type EDIT from Logo, without any procedure name following the command, and you will be returned to the Logo editor.

TEACHING SAM TO DO THINGS WITH ONE KEYSTROKE

Logo will allow you to program the keyboard so that you can do things with only one keystroke.



Here is a time-saving idea! Are you bored with having to type CATALOG every time you want to see what is on your disk? This procedure will teach Sam to fetch the catalog for you when you simply type the letter C.

Here is the procedure.

```
TO C
CATALOG
END
```

It is a short procedure, but a very useful one. You can teach Sam all sorts of similar tricks using similar procedures! How about writing a procedure that will allow you to edit the procedure BOX with one keystroke? It would look like this.

```
TO E
EDIT "BOX
END
```

Typing E is now all that would be needed to put Logo into its edit mode with BOX in its edit buffer. Can you think of any other keystrokes that you might want to abbreviate?

MAKING SAM WAIT

When you call for a catalog, have you found that there are more lines in the catalog listing than there are on the screen? If there are more than 20 lines in the catalog listing, then the first lines will be impossible to read. To freeze the lines that you want to look at, use CTRL-W. The W stands for Wait. When you are ready to continue, press any key. CTRL-W can be used in other Logo modes, not just the edit mode. Experiment with CTRL-W to see what it can do for you!

It is only practice that will make the editor and its commands seem familiar. The more familiar you are with them, the more useful they will be. You should practice editing procedures by trying some of the challenges that follow the summary of this chapter.

SUMMARY

Here is a summary of the topics and keystrokes that we covered in Chapter 4.

1. EDIT "SQUARE will put Logo into the edit mode to begin editing the procedure SQUARE.
2. When in the edit mode, you can use the control key to perform deletions, insertions, and replacements.
3. The most common control keys are:

left arrow	erase characters
CTRL-A	Move to the beginning of the line
CTRL-B	backspace
CTRL-C	Exit from the editor
CTRL-D	Erase the character under the cursor
CTRL-E	Move the cursor to the end of the line
CTRL-F	Move the cursor forward
CTRL-G	Move to Logo, and cancel all changes
CTRL-K	Move text to the kill buffer
CTRL-N	Move to the next line
CTRL-P	Move to the previous line
CTRL-Y	Yank text from the buffer
[RETURN]	Move the cursor and the following text to the beginning of the next line

4. ERPS will erase all of Sam's procedures.
5. ER "BOX will erase only the procedure BOX.
6. POTS will print out the titles of the procedures that Sam knows about.
7. CTRL-T will give you a full screen of text.
8. CTRL-S will give you a split screen containing both text and graphics.
9. CTRL-L will give you a full screen of graphics.
10. In Logo, you can use a procedure to allow single key control of the turtle.
11. CTRL-W will tell Logo to wait until another key is pressed.
12. There are three ways to place a procedure in Sam's memory, they are
    - by using the TO command.
    - by using the editor and the name of the procedure.
    - by loading an existing procedure from disk.

CHALLENGES

1. Use the editor to change the first procedure into the second procedure.

Change

```
TO SIX.SQUARE
REPEAT 6 [SQUARE]
END
```

to

```
TO FOUR.SQUARE
REPEAT 4 [SQUARE]
END
```

2. Change

```
TO EIGHTSQUARE
REPEAT 4 [SQUARE]
LT 50
REPEAT 4 [SQUARE]
END

TO SQUARE
FD100
RT90
FD100
RT 900
FD 100
RT 90
FD1000
END
```

to

```
TO EIGHT.SQUARE
REPEAT 4 [SQUARE]
LT 45
REPEAT 4 [SQUARE]
END

TO SQUARE
FD 100
RT 90
FD 100
RT 90
FD 100
RT 90
FD 100
END
```

3. Change

```
TO WINDMILL
RPEAT 6 [SKINNYRECTANGLE]
TO SKINNY.RECTANGLE
RT 90
FD20
RT 90
FD 100
```

to

```
TO WINDMILL
REPEAT 4 [SKINNY.
RECTANGLE]
END

TO SKINNY.RECTANGLE
FD 100
RT 90
FD 20
RT 90
FD 100
RT 90
FD 20
RT 90
END
```

QUIZ

1. Typing EDIT "SQUARE. will put Logo into the edit mode, to begin editing the procedure named _____.
2. When in the edit mode, you use the control keys to perform _____, insertions, and _____.
3. The most common control keys are:

back arrow erase character

CTRL-A	_____
CTRL-B	Backspace
CTRL-C	_____
CTRL-D	Erase character under the cursor
CTRL-E	Move the cursor to the end of the line
CTRL-F	_____
CTRL-G	Move to Logo, and cancel all changes

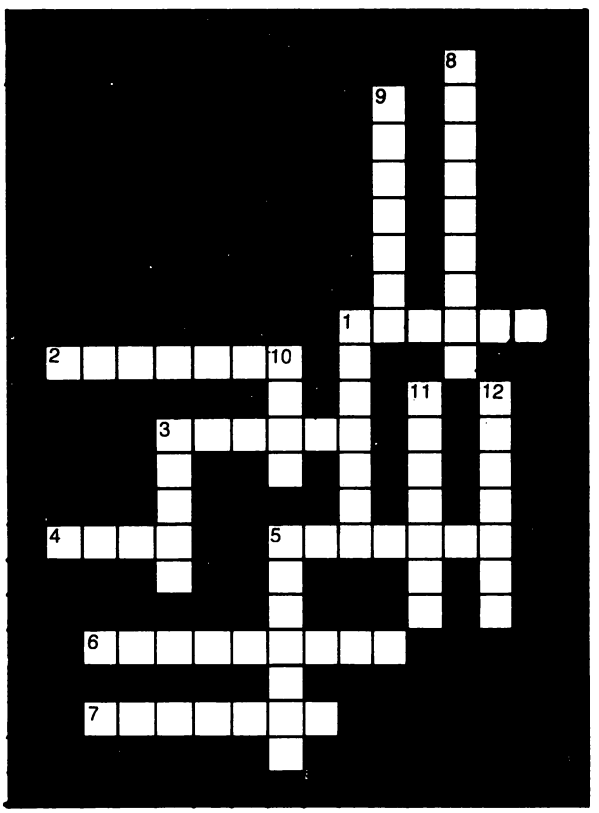
CTRL-K Move text to the kill buffer
CTRL-N _____
CTRL-P Move to the previous line
CTRL-Y _____
[RETURN] Move the cursor and the following text to the beginning of the next line

4. ERPS will _____ all of Sam's procedures.
5. ER "BOX will erase only the _____.
6. _____ will print out the titles of the procedures that Sam knows about.
7. _____ will give you a full screen of text.
8. CTRL-S will give you a _____ of text and graphics.
9. _____ will give you a full screen of graphics.
10. In Logo, you can use a procedure to allow _____ control of the turtle.
11. CTRL-W will tell Logo to _____ until another key is pressed.
12. There are three ways to define a procedure in Logo, they are.
    - By using the _____ command.
    - By using the _____ and the name of the procedure.
    - By _____ an existing procedure from disk.

CROSSWORD PUZZLE

Clues Across

1. Something that will give you a splitscreen.

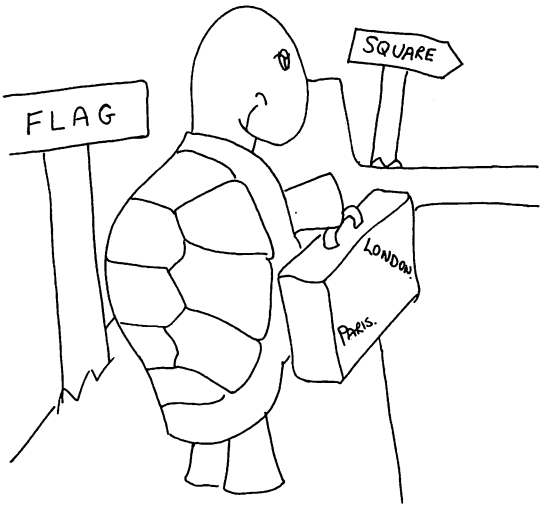


2. This will send you back a line.
3. This will save us a lot of typing.
4. If you want to see what Sam knows, type this.
5. Something that will give you a full graphics screen.
6. Be careful! This will rub things out!
7. Open up, and let me in!

Clues Down

8. Type this to edit procedure SQUARE.
9. When your procedure is complete, type this.
1. To look at the text on the screen.
10. If you want to see the names of the things that Sam knows, type this.
11. Move right back please!
12. This will move you to the beginning of a line.
3. ER stands for rub me out?
5. Next please!

Chapter 5



Using Color with Sam

Now that you know how to edit, it will be much easier for you to make changes to our procedures. This means that we can look at some more primitives that are used in Logo.

Let's look at some of the colorful things that we can do with Sam! Sam can use a lot of different colors for both his background and his pen. It is very easy to change the colors in Logo. Let's start by looking at the different colors that you can use.

COLORS THAT YOU CAN USE

There are six colors that you can use in all, but Sam doesn't understand colors in the same way that we understand colors. Being a computer, he only truly understands numbers. For Sam, the number 2 represents the color green, and the number three represents the color violet. Here is a complete list of colors and the numbers that are used to represent them in Logo.

0 black
1 white
2 green
3 violet
4 orange
5 blue

Up to now, we have simply used the black of the screen as a background. Now that we know the colors that we can use and the numbers that represent them, let's look at how to change Sam's background color! I wonder what the background will look like if we use some of the colors listed?

CHOOSING THE BACKGROUND COLOR

Let's choose a nice background color for Sam! How about green? Turtles like green a lot, I suppose its because they like grass! To set the background color to green, type in the following:

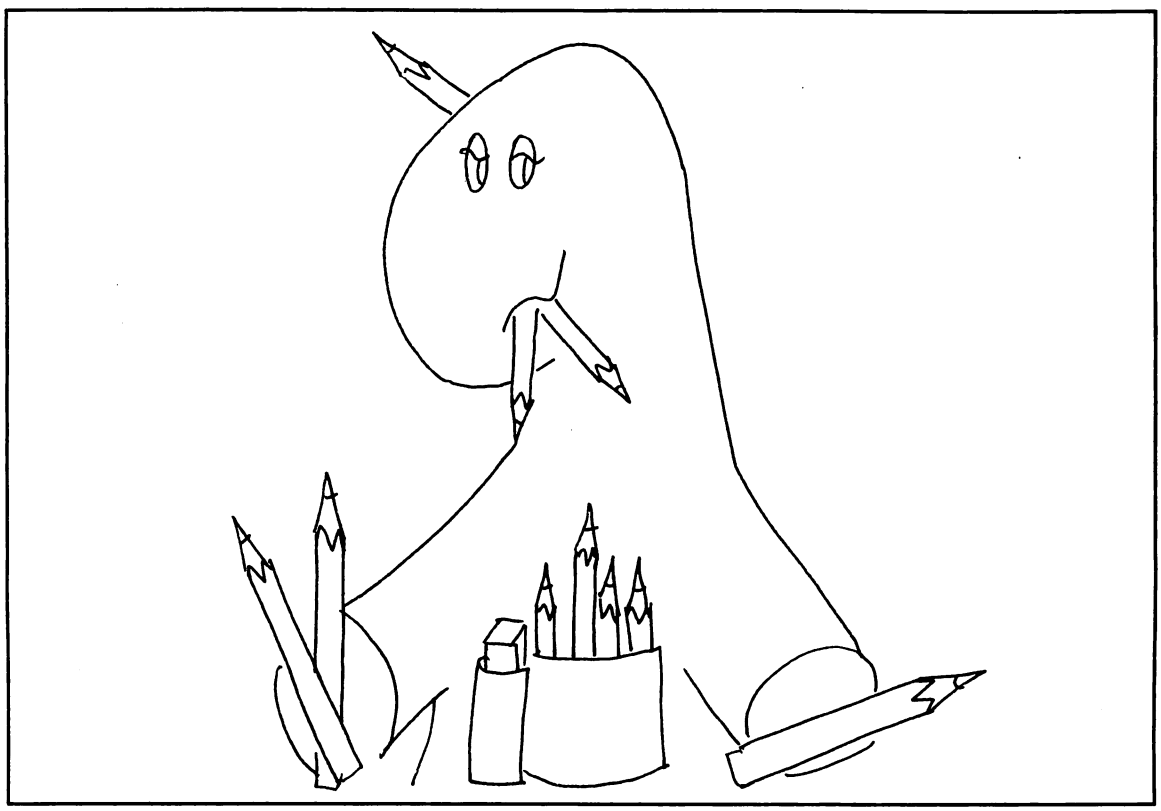
```
SETBG 2
```

SETBG stands for SETBackGround, and the 2 stands for green, of course. The screen turned a nice shade of green—just what we wanted! Any time you want to change the background color, simply use the SETBG primitive along with the number that represents the color that you want.

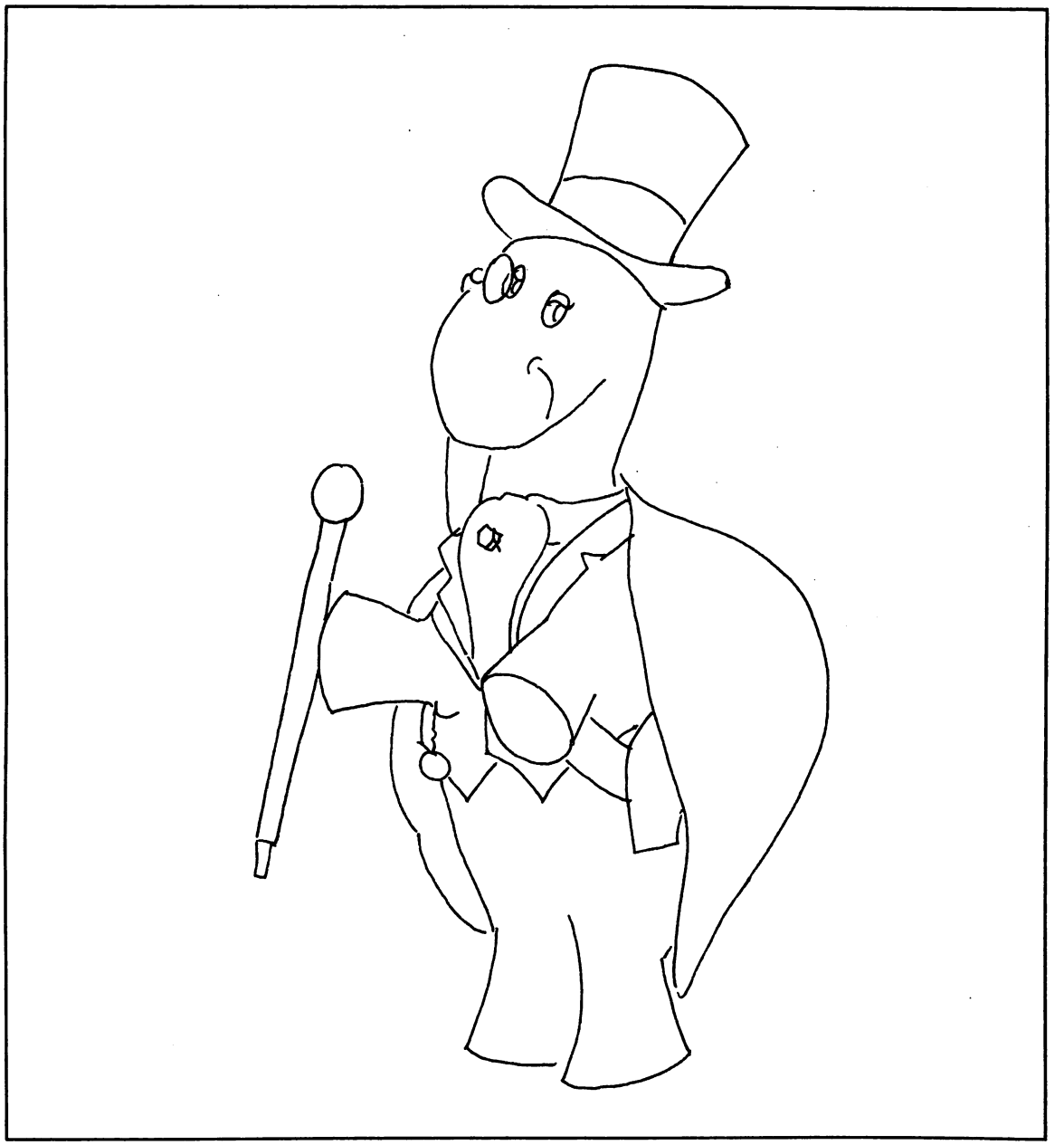
Here is a procedure that will demonstrate the colors that you can use for Sam's background.

```
TO COLOR
SETBG 5
BOX
SETBG 4
BOX
SETBG 3
BOX
SETBG 2
BOX
SETBG 1
BOX
SETBG 0
BOX
END

TO BOX
CS
REPEAT 4 [FD 100 RT 90]
END
```

When you finish typing this procedure in, type COLOR to see what the procedure does. Would you like to try your own program



Sam has a lot of pens and colors that he can use!



Sam likes to be elegant.

with changing colors in it? How about trying a procedure from one of the other chapters, but this time you could program Sam to use colors! If either of these suggestions appeal to you, then please feel free to try them!

In the Apple Logo manual, you will find that there is a sixth background color listed. The color is black, for use with a black and white monitor. To keep things simple, we have ignored this sixth color up to now.

It was easy to change the background color, I wonder if it is as easy to change Sam's pen color?

CHOOSING THE PEN COLOR

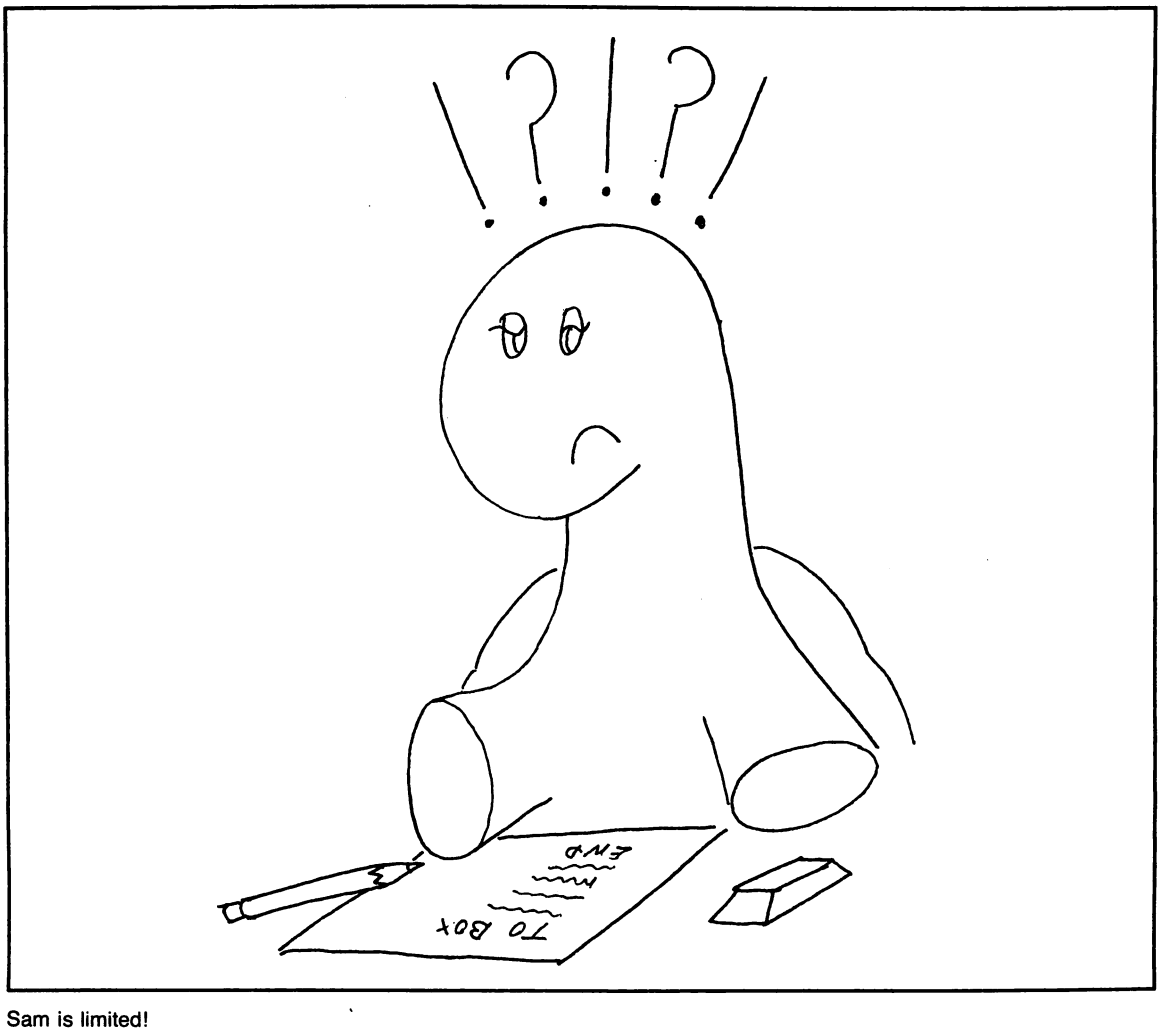
What does Sam use to draw his diagrams on the screen? He uses a *pen*! It is a *programmable pen* of course. After all Sam isn't just a turtle, he is a computer turtle. I think a computer turtle should have an electronic programmable multicolored pen, don't you?

You can change the pen color to any of the six colors that have already been listed. The primitive SETPC will give you control of Sam's pen. SETPC stands for SETPenColor. SETPC also needs a number to tell Sam what color of ink to use. If you type the following

```
SETPC 2
```

you will find that the pencolor will be set to green.

Of course, I don't have to tell you that it



Sam is limited!

wouldn't be a good idea to use a green pen on a green background, because, suddenly, Sam's track would disappear completely!

Here is a challenge for you. Can you draw a blue square on a white background? You should try this for yourself. Use the SETPC and SETBG primitives from this chapter. The procedure for drawing a square was given earlier.

Two possible solutions are given below, there are of course a number of ways of solving this problem.

```
TO BLUESQUARE
CS
SETBG 1
SETPC 5
REPEAT 4 [FD 50 RT 90]
END
```

Or you could have used this procedure.

```
TO BLUESQUARE 2
CS
SETBG 1
SETPC 5
SQUARE
END

TO SQUARE
REPEAT 4 [FD 50 RT 90]
END
```

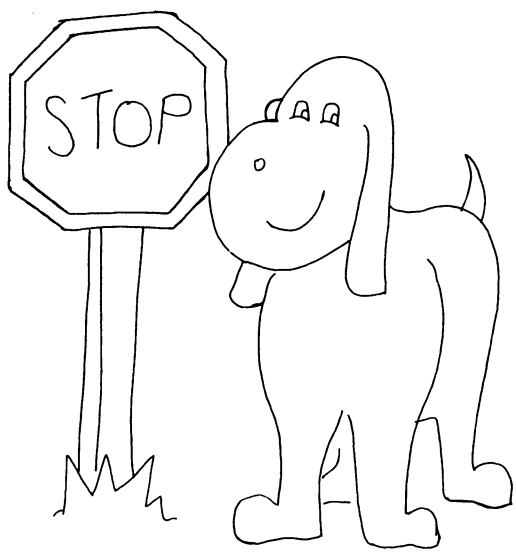
Both procedures will give the same result, but the second one is more *elegant* because it breaks the procedure into subprocedures. Sam likes to be elegant!

Let us look at this program a little more closely. Here is what we might have said in English about drawing our blue square.

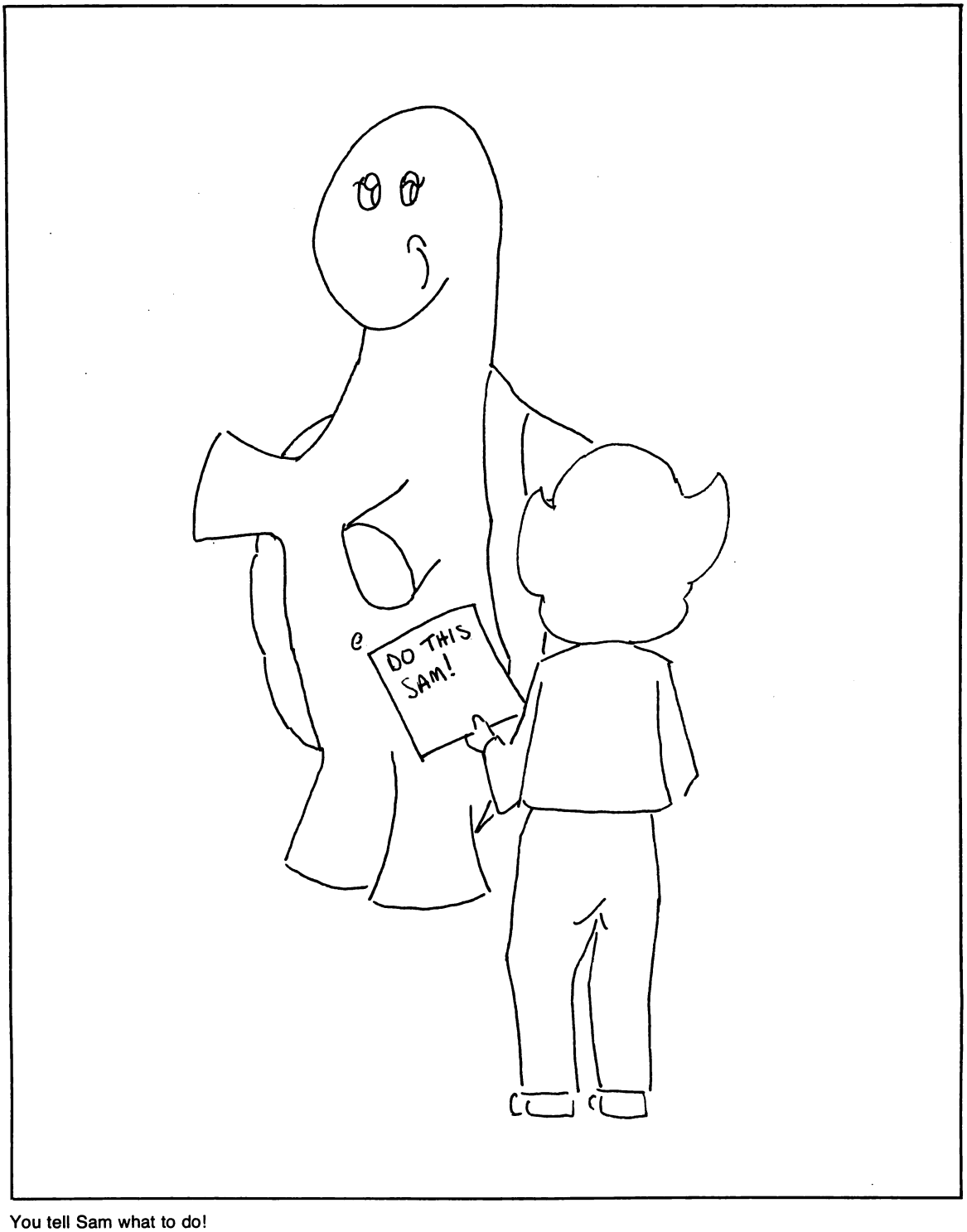
```
Clear the screen
Set the background color
Set the pen color
Draw the square
End
```

As we said before, it is often a good idea to use these kinds of English lists to help you to solve a problem.

Just a word of warning, using colors on a color monitor or television set can sometimes give unpredictable results. It depends on the color combinations that you use. The unpredictable results occur because of the way that a color tube and the Apple computer works. This problem is a problem that cannot be solved very easily! Like any other machine, the computer system is not perfect. Sometimes you have to live with the limitations of your system.



A chart that will explain some of the strange color-effects that you can get with Sam follows.



You tell Sam what to do!

Table 5-1. A Chart of the Strange Color Effects You Can Get with Sam.

Pen Color Number	Background Color Number 0	1	2	3	4	5	6
0	BLACK	BLACK	BLACK	BLACK	BLACK	BLACK	BLACK
1	WHITE	WHITE	WHITE	WHITE	WHITE	WHITE	WHITE
2	GREEN	GREEN	GREEN	GREEN	ORANGE	ORANGE	WHITE
3	VIOLET	VIOLET	VIOLET	VIOLET	BLUE	BLUE	WHITE
4	ORANGE	ORANGE	ORANGE	GREEN	GREEN	ORANGE	ORANGE
5	BLUE	BLUE	VIOLET	VIOLET	BLUE	BLUE	WHITE

The strange results are circled, you can avoid these combinations if you like, or you can use them for special effects if that is what you want to do.

Here is another procedure for you to try.

```
TO RANDOM.COLORS
REPEAT 100 [SETBG RANDOM 6
SQUARE]
END

TO SQUARE
REPEAT 4 [FD 50 RT 90]
END
```

There is something new in the procedure, the primitive RANDOM is a primitive that you have not met before. RANDOM 6 will allow Logo to choose a color from zero to five at random. *At random* is something like rolling a die; each number on the die has an equal chance of being on the top. The procedure will use the random number from the RANDOM primitive to select the pen color; the pen colors, of course, run from 0 to 5. Type RANDOM.COLORS to see the demonstration.

It isn't very convenient to always have Sam drawing when he moves. If his pen is down all the time, you might find that you get lots of lines that you didn't really want. In fact, sometimes Sam will make quite a mess if you simply allow him to draw every time he moves.

Let's look at how to control Sam's pen some more!

PEN UP AND PEN DOWN

Sam can decide, pardon me, I should have said that *you* can decide, whether or not Sam will draw. That is, you can tell Sam to lift his pen or to lower his pen. This is called *penup* and *pendown*. Here are the primitives that you need to raise and lower Sam's pen.

```
PU
PD
```

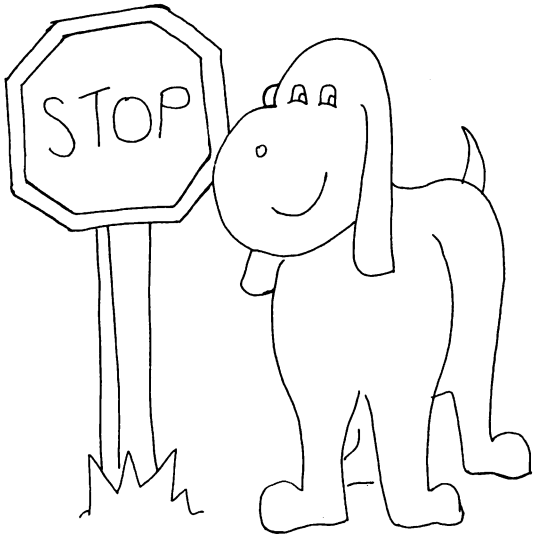
PU stands for PenUp and PD stands for PenDown. Did you notice that PU and PD do not have any numbers following them?

If the pen is up, Sam will leave no marks as he moves about on the screen. If Sam's pen is down, he will leave a mark on the screen as he moves around. The marks will be the color that you last gave to Sam's pen.

Here is a program for you to type in. It will show you how you can use PU and PD.

```
TO DOUBLE.SQUARE
SETBG 2
SETPC 0
SQUARE
```

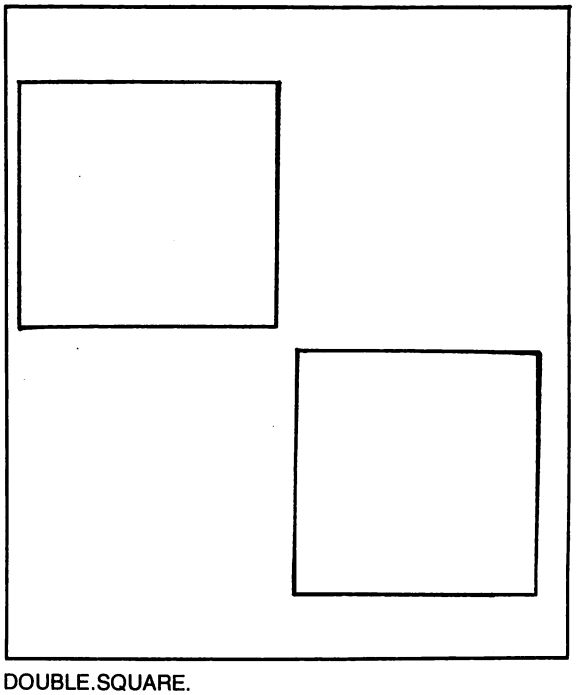
```
PU
FD 50
LT 90
FD 60
RT 90
PD
SQUARE
END

TO SQUARE
REPEAT 4 [FD 50 RT 90]
END
```



Remember to use those English statement lists!!!

Here is the program written in English statements.

Set the background color
Set the pen color
Draw a square
Lift the pen
Move a little
Put the pen down
Draw another square
End



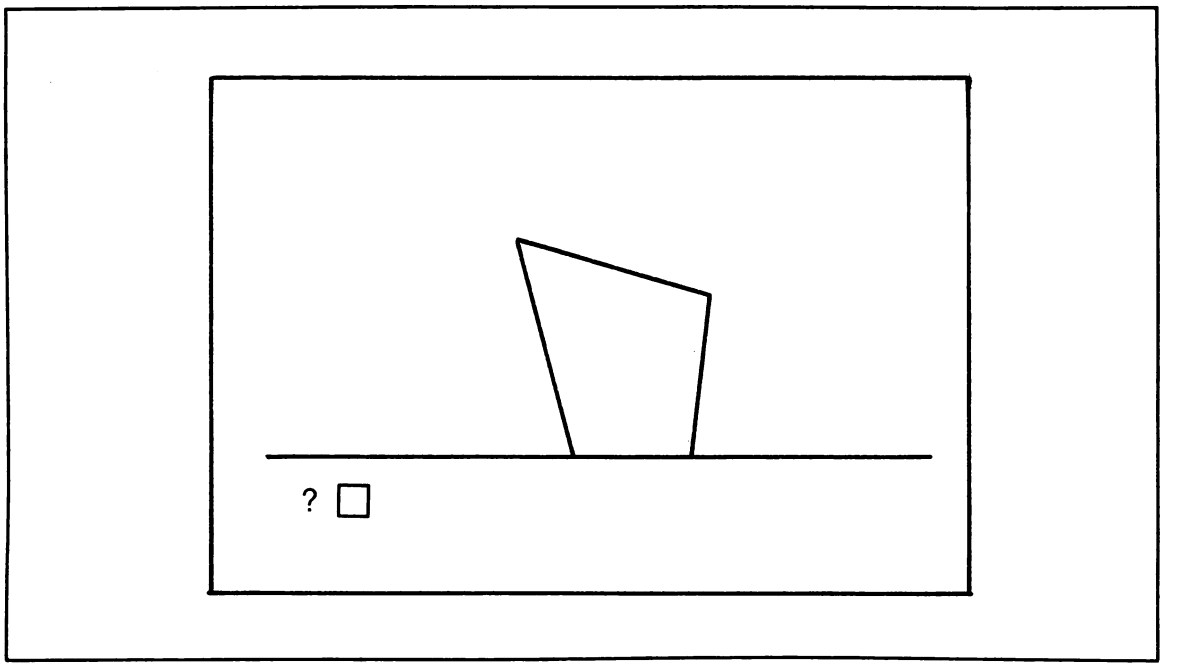
DOUBLE.SQUARE.

Type DOUBLE.SQUARE to see what the program will do.

CHANGING SAM'S SCREEN

You now know how to control Sam's pen and his background. Is there anything else that you can change in Sam's world? Yes, there is! You can control his screen! You can change Sam's screen and the way that it operates. In fact, you can forget all about Sam if you want to, and look only at text on the screen. If you read Chapter 4 carefully, you will remember that you learned how to change the screen by pressing keys. Now you will learn how to use primitives that control the screen within your procedures.

You will have noticed of course, that you do not have the use of all of the screen for Sam. The bottom four lines are reserved for text, so that you can see the primitives and procedures that you are typing. Here is an illustration of the screen.



This is the splitscreen mode.

Do you remember that when we talked about controlling the screen format from the keyboard, we used the CTRL keys. Do you remember that CTRL-S will give you the splitscreen and CTRL-T will give the textscreen. Ah yes. . . and CTRL-L will give you the fullscreen. We can control the screen format from procedures as well as from the keyboard. This means that Sam's screen is *programmable.*

It is really quite difficult to tell Sam what to do when you can't see the things you are typing or trying to type! However it is sometimes better for us to give Sam the whole screen to himself. This is done by including the following primitive in your procedure.

FULLSCREEN

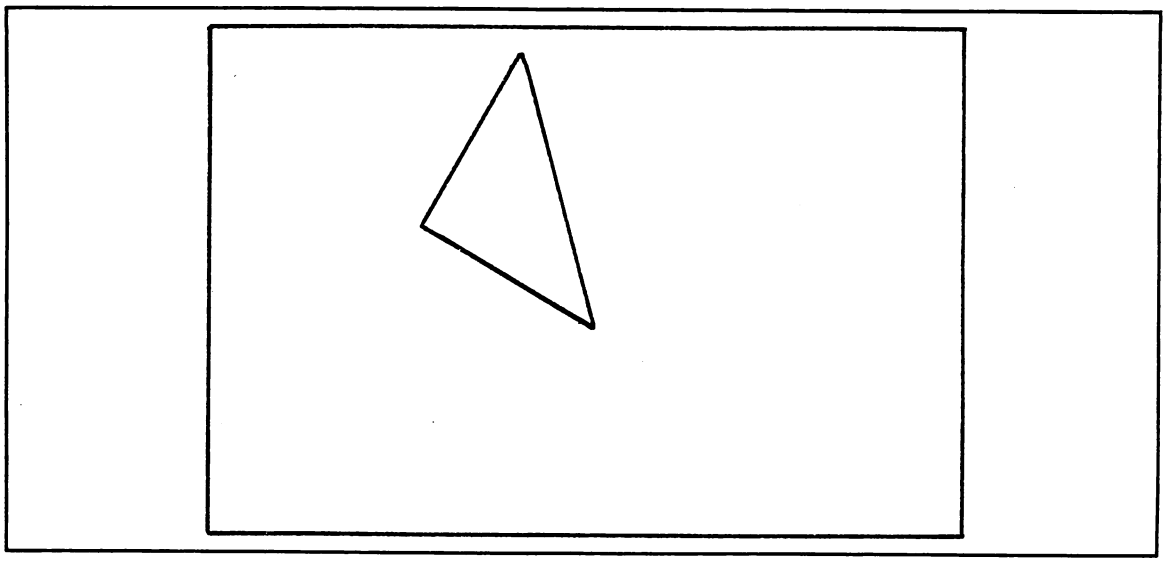
There is no abbreviation for FULLSCREEN and there are no numbers or values to follow it either. The fullscreen is shown on page 92. This screen mode will give you more room for Sam and his procedures. After you have used the fullscreen in a procedure, you can return to the splitscreen mode from the keyboard by typing CRTL-S.

If you want to go back to the splitscreen from a procedure, not from the keyboard, you simply use the following primitive in your procedure.

SPLITSCREEN

What about the textscreen mode? You can move Sam into his textscreen mode by using the primitive command TEXTSCREEN. When you use the textscreen mode, Sam will disappear, and only text will show on the screen. Sam is still there, but you cannot see him or what he is doing from the textscreen!

Here is a short program that will demonstrate these commands.



This is the fullscreen mode.

```
TO SCREEN.MODE.DEMO
PR [HERE COMES THE FULL-
SCREEN MODE!]
WAIT 120
SETBG 3
FULLSCREEN
WAIT 120
SQUARE
WAIT 120
SETBG 2
SPLITSCREEN
PR [HERE IS WHERE THE TEXT
IN THE SPLITSCREEN MODE
WILL APPEAR!]
WAIT 120
TEXTSCREEN
WAIT 120
PR [WE ARE NOW IN THE TEXT-
SCREEN MODE!]
END

TO SQUARE
CS
REPEAT 4 [FD 100 RT 90]
END
```

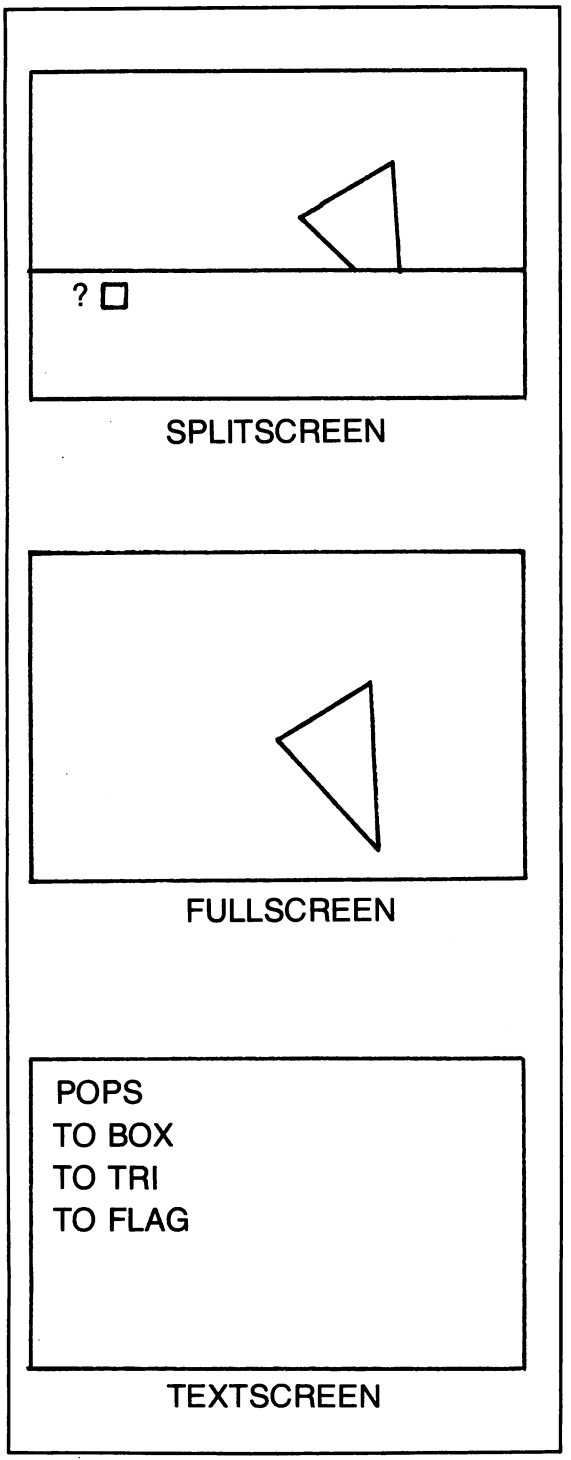
Type SCREEN.MODE.DEMO to see how the screen behaves in its different modes.

Just a minute, there is something new here that we have not seen before! What does the primitive WAIT do, and what does the number 120 mean?

WAIT will tell the computer to wait for a given number of seconds, so that WAIT 120 will make the computer wait for a total of 2 seconds. If you used WAIT 60 instead of WAIT 120, you would find that the computer would wait for exactly one second.

You use the WAIT primitive if you want to slow the computer down. As you know, computers can do things very quickly! Sometimes, they do things so quickly that it is difficult to see what they have done! For instance, if you ask Sam to change his background color, he will do so instantly. To keep the color on the screen for a while before the next color change takes place, you will probably need to use the WAIT primitive.

If you have told Sam to do something that already takes time to do, you will not have to use the WAIT primitive in your procedure. For



This is what the screen modes look like.

example, if you have asked Sam to draw a square in your procedure, he will probably take a second or two to complete it. If it does take a second or two, it is unlikely that you will need a WAIT primitive!

Why not remove the WAIT primitives from the SCREEN.DEMO procedure to see if you really needed them!

The other thing to notice is the PR command. It stands for PRint, and it will send the text in the brackets to the video display. This means that you will be able to include messages in your Logo procedures.

Let's see if there are any other Logo commands that will control Sam's screen.

Using Wraparound and Window

Do you remember what wraparound is? It is when Sam moves off the edge of the screen and reappears on the other side! Sometimes, you will not want to have the turtle wraparound. If you have wraparound, you often get an overlapping of Sams patterns. This may make a pattern look better, but it may also give a very messy diagram!

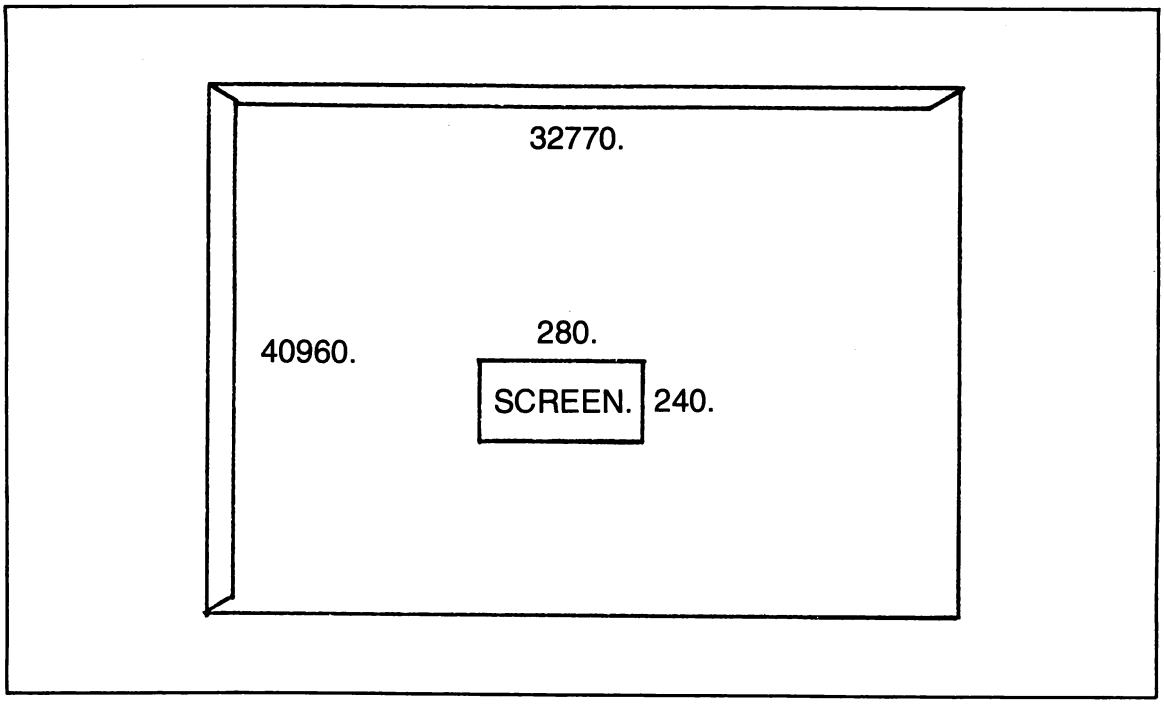
Logo gives you a number of commands that can control wraparound on the screen. Here is one of them. Type the command

```
WINDOW
```

This command will stop the wraparound. After you have typed the WINDOW command, type the following:

```
CS
FD 200
RT 160
FD 200
```

What happened? Sam moved off the screen and disappeared. He didn't reappear on the bottom of the screen as he usually does! So



A map of Sam's universe in turtle steps.

where did he go? He hasn't really gone away of course, but he is outside the screen's *window*. It is possible to tell Sam to go for quite a distance before he reaches the absolute edge of his *world*. His world is fairly large, so that we usually will not have to worry too much about him falling off the edge of the planet!

Can you find out how large Sam's world really is? Try sending Sam on longer and longer trips! What did Sam say when he reached the edge of his universe?

As I mentioned before, the advantage of using the WINDOW primitive is that Sam will not reappear on the opposite edge of the screen and draw on top of himself. You should use WINDOW a lot in your procedures. Using WINDOW will *clean up* your diagrams for you. If you have complicated diagrams that look like spaghetti, then WINDOW is a good primitive for you to try!

However, if you do use window, you must be careful that you don't "lose" poor Sam, he does depend on you to take care of him after all! With WINDOW, you might find that if Sam is out of sight, he might also be out of mind.

Of course you will want to go back to wraparound eventually. To do this, simply type the primitive

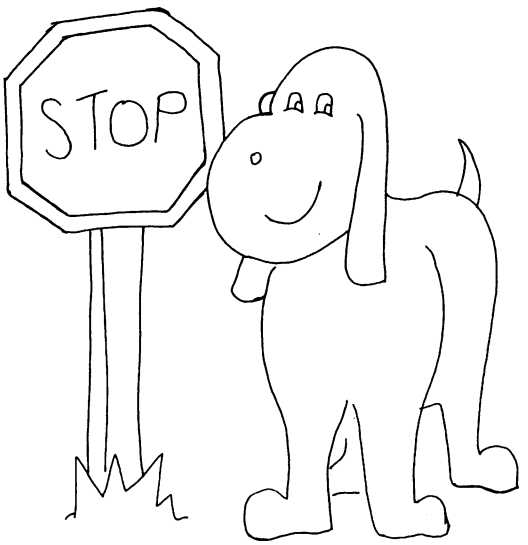
```
WRAP
```

or include it in the appropriate place in your procedure.

There is one other screen command that you might find useful in your procedures. If you type

```
FENCE
```

you will find that Sam will be fenced in by the edges of the video.
Be careful of the primitive FENCE!

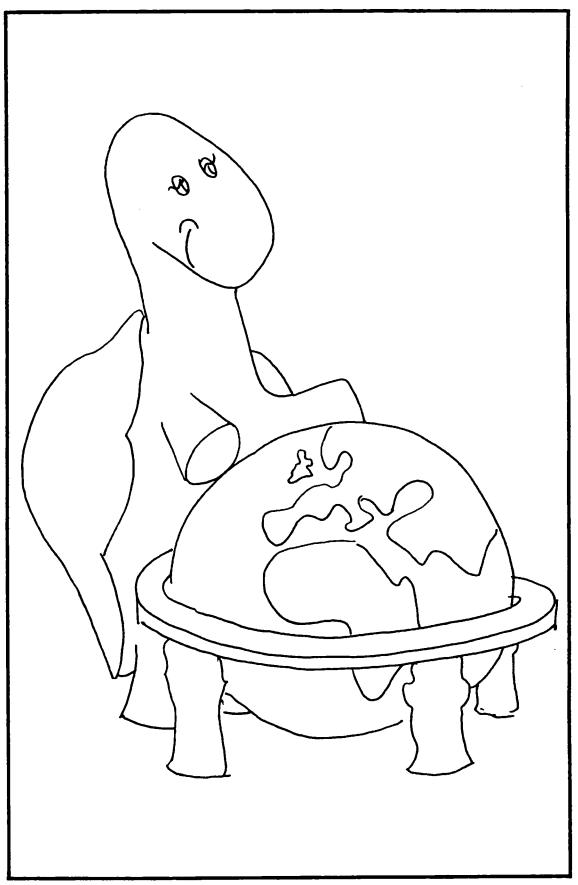


If Sam does bump into his fence, Logo will let you know by stopping the program with an error message. As you can imagine, this could be inconvenient. You must use FENCE with care.

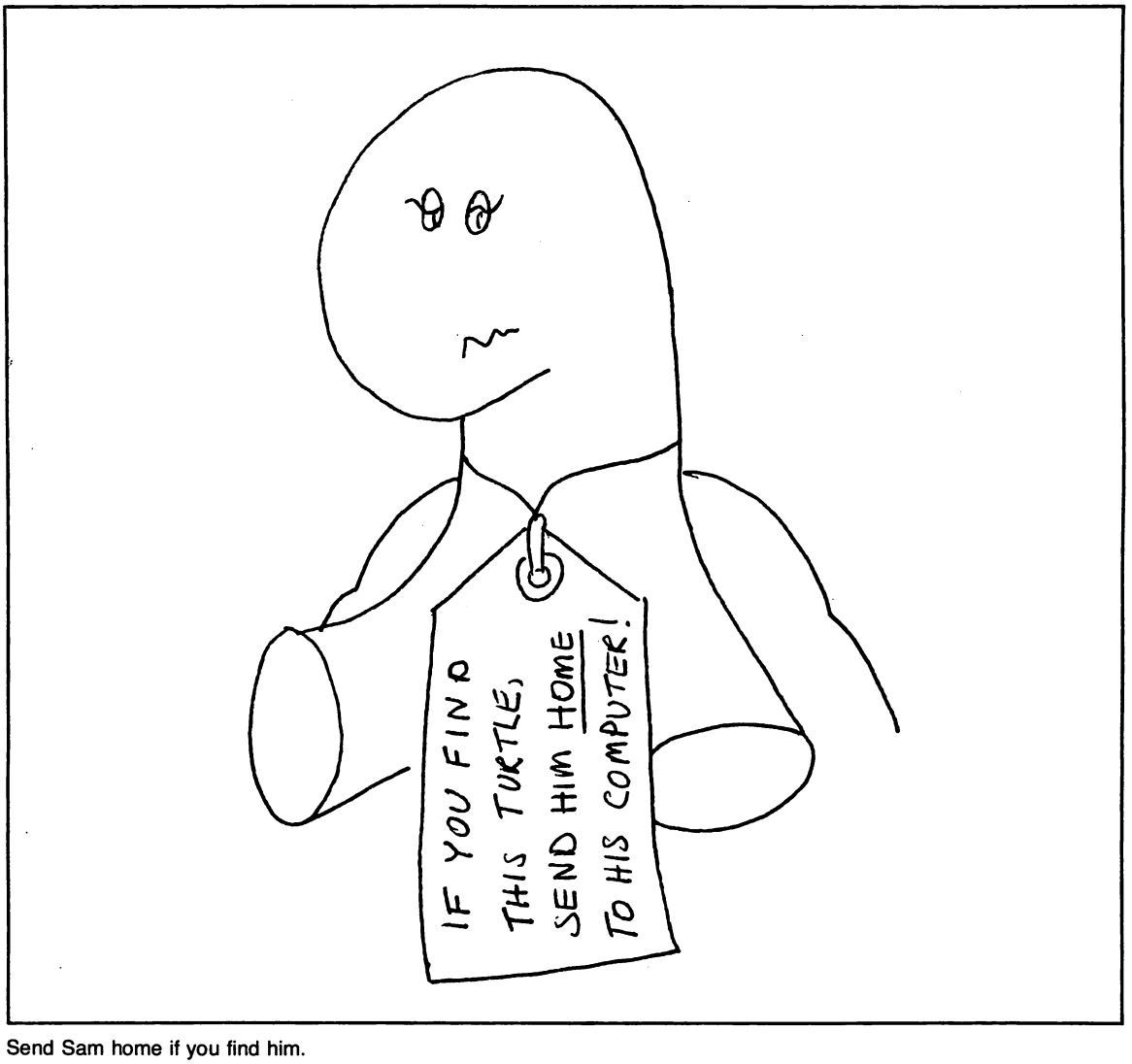
Here is a procedure for you to type in that will illustrate what happens when you use the screen control commands.

```
TO SCREEN.WRAP.DEMO
WRAP
SETBG 2
PR [THIS IS WRAP MODE!]
WAIT 120
BIGMOVE
WAIT 120
WINDOW
PR [THIS IS WINDOW SCREEN
MODE]
WAIT 120
BIGMOVE
WAIT 120
PR [WATCH FOR SAM'S MES-
SAGE ON THE FENCE MODE
WAIT 120
CS
FENCE
BIGMOVE
END

TO BIGMOVE
CS
RT 40
FD 3000
END
```

Unfortunately, as we said before, if Sam bumps into the fence, he will complain about it. That is maybe quite fair. After all, Sam can't see very well, and how would you feel if some-



Sam and his world.



Send Sam home if you find him.

body gave you some instructions that allowed you to walk into a fence? Will Sam bump into the fence in our SCREEN.WRAP.DEMO procedure? Type SCREEN.WRAP.DEMO to see Sam do his stuff and find out if he bumps the fence at the end of the procedure.

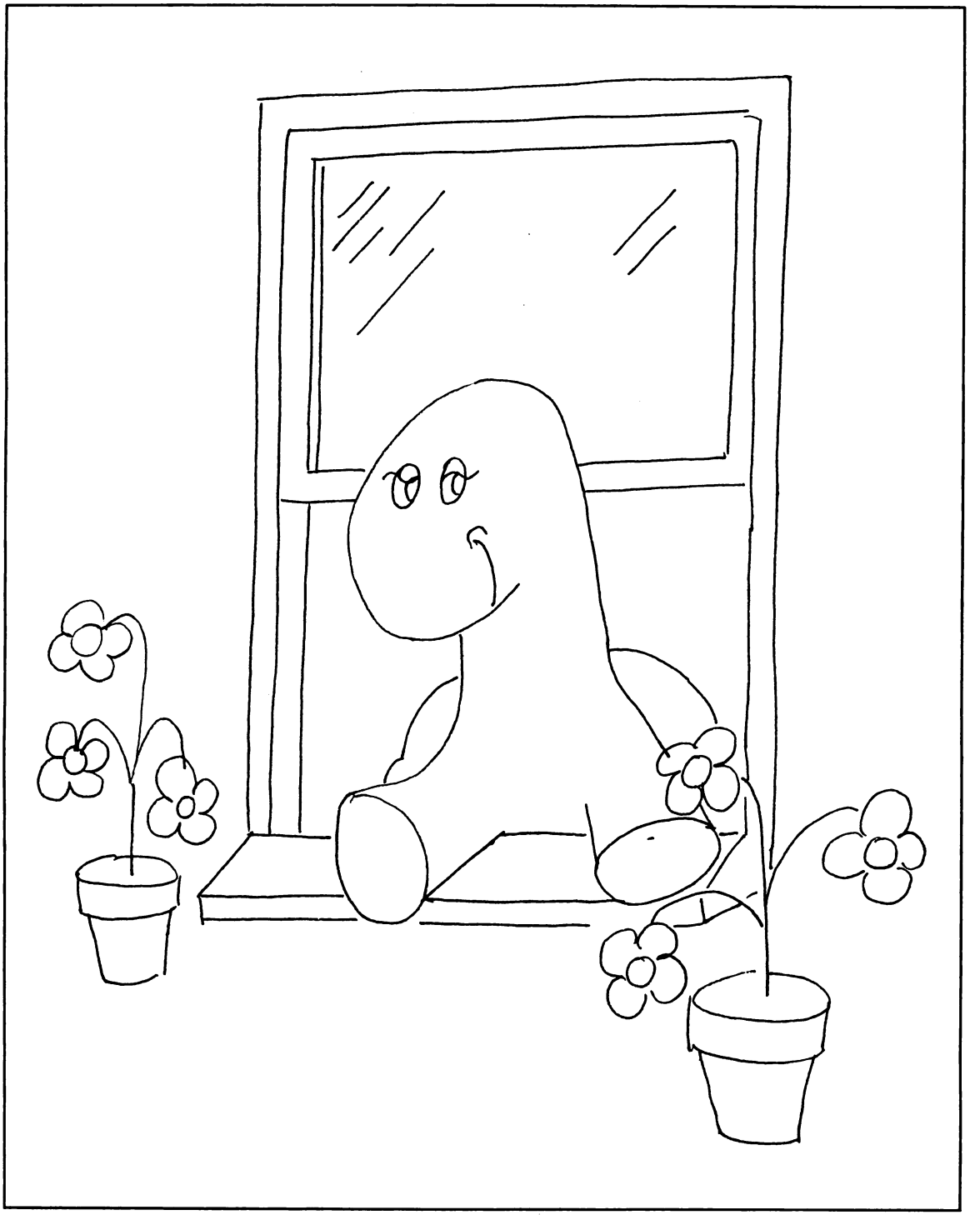
SUMMARY

Here is a summary of the new ideas and primitives that we met for the first time in Chapter 5.

1. The colors that we have available are:

0-black	3-violet
1-white	4-orange
2-green	5-blue

2. To set the background color, use SETBG 3 (for violet).



Sam's window.

3. To set the pencolor to orange, you would type SETPC 4.
4. Sam, like all computer systems is limited. Sometimes it is difficult to overcome these limitations.
5. To raise the pen use the PU command.
6. To lower the pen use the PD command, (no numbers are needed).
7. To use all of the screen for turtle, use the FULLSCREEN command.
8. To use the turtle and text on the same screen use the SPLITSCREEN command.
9. To eliminate wraparound use the WINDOW command.
10. To go back to wraparound use the WRAP command.
11. To keep the turtle within the bounds of the screen, use the FENCE command.

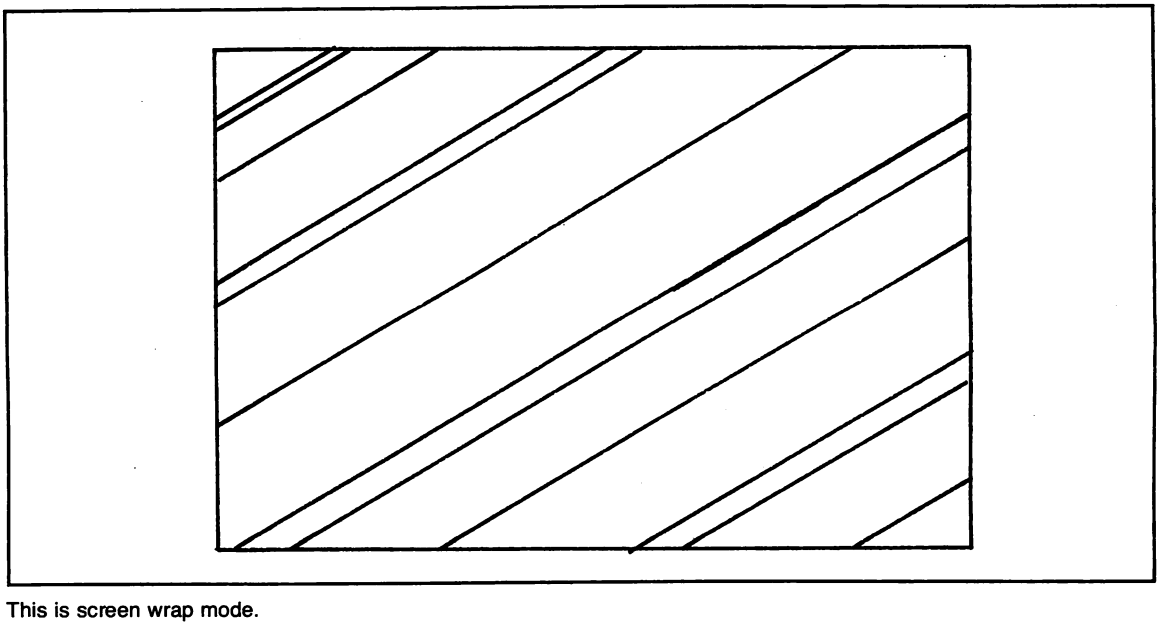
CHALLENGES

1. Using a procedure called HOUSE, draw a house and then change the background color of the house. Start with black and go through all the colors until you get to blue. Use six different backgrounds in all.
2. Using a procedure called HOUSE, draw a house with green as the background and then redraw the house using different pen colors. Have the procedure use all the pen colors available in turn. Start with black and end with blue.
3. Using PU and PD, draw two houses on a blue background with a black pen.
4. Using all of the screen draw a white flower on a blue background.
5. Draw a very large black flower on a white background. Make sure that the flower does not wraparound.
6. Write a procedure called FLASH that will draw a pattern and then flash different backgrounds for the pattern.

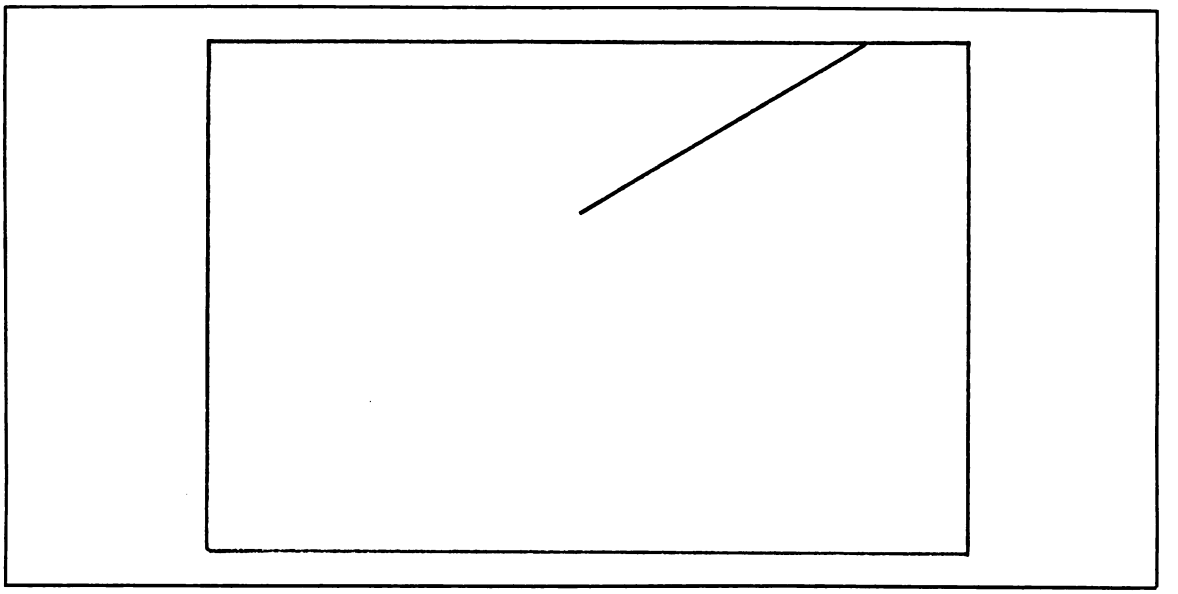
QUIZ

1. The colors that we have available are:

   0-____
   1-white



This is screen wrap mode.



This is window screen mode.

2-____
3-violet
4-____
5-blue

2. To set the background color, use _____ 3 for violet.
3. To set the _____ to orange, you would type SETPC 4.
4. Sam, like all computer systems is _____. Sometimes it is difficult to overcome these limitations.
5. To raise the pen use the _____ command.
6. To _____ the pen use the PD command, (no numbers are needed).
7. To use all of the screen for turtle, use the _____ command.
8. To use the turtle and text on the same screen use _____.
9. To eliminate _____ use the WINDOW command.
10. To go back to wraparound use the _____ command.
11. To keep the turtle within the bounds of the screen, use the _____ command.

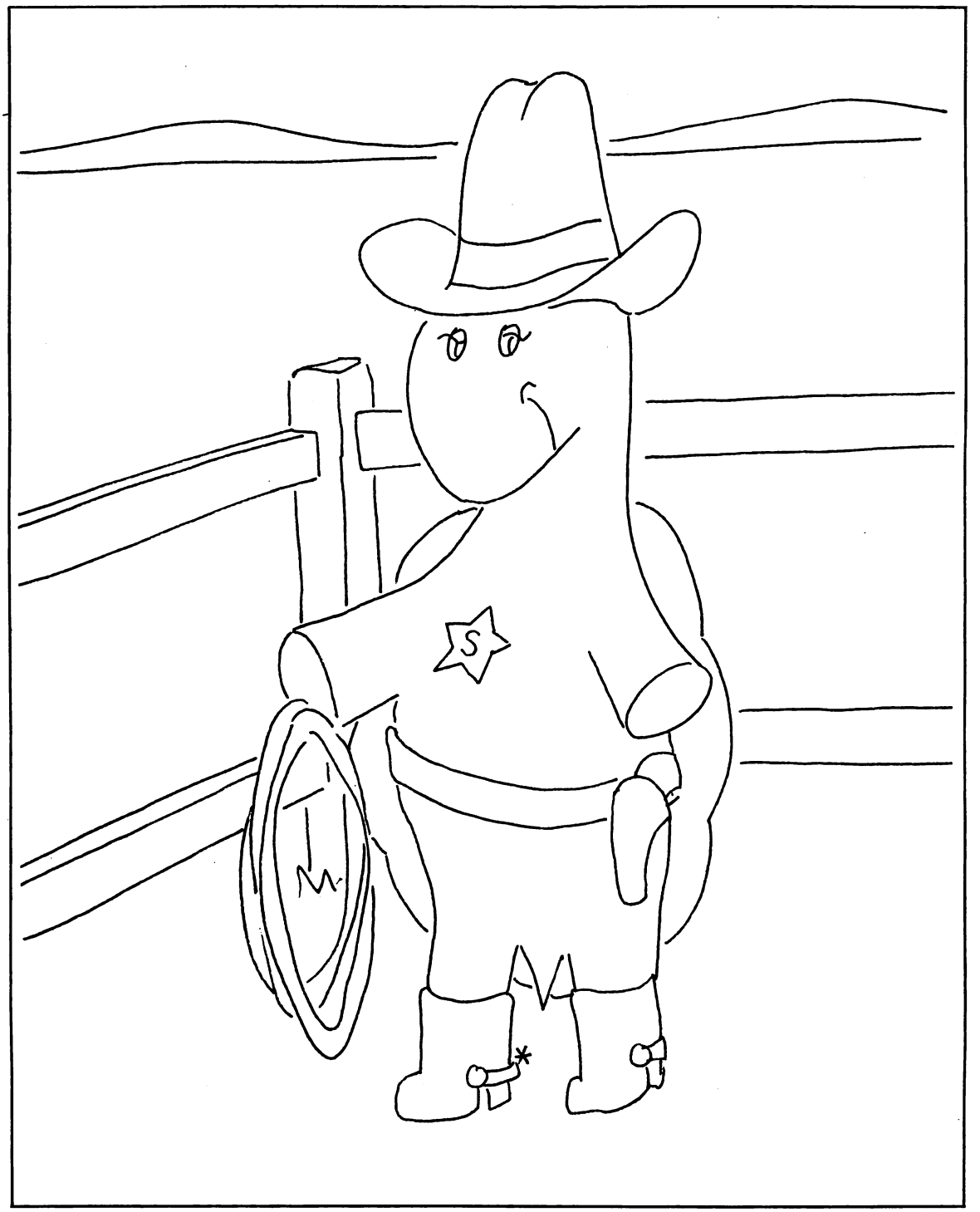
CROSSWORD PUZZLE

Clues Across

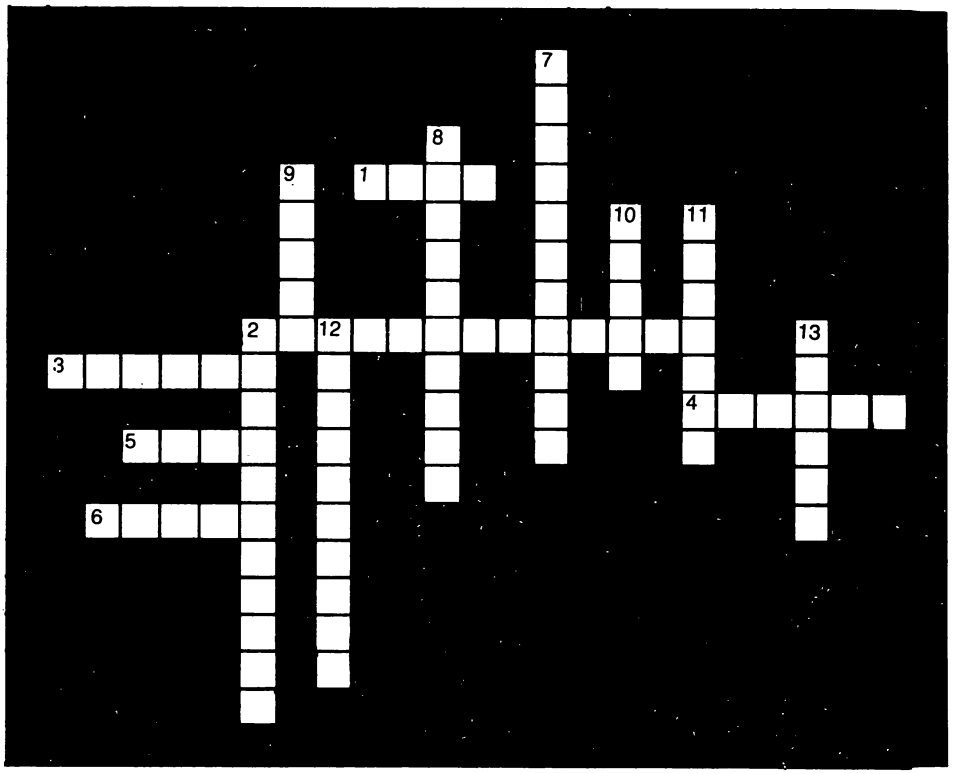
1. 5 is the number for this color!
2. Use this primitive to color your screen.
3. 4 is the number for this color!
4. You can look through this.
5. Sam will appear on the other edge if you use this one!
6. 2 is the number to use for this color!

Clues Down

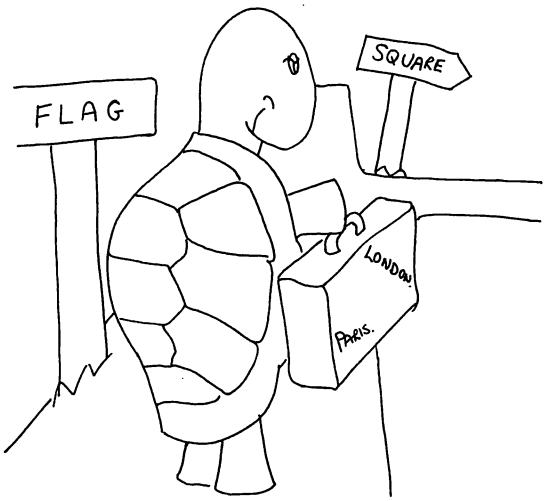
7. Text and video are shown together on this.
8. Only Sam appears on this one.
9. If you use this primitive, Sam will bump the edges.
10. This will stop Sam drawing.
11. This will help Sam to draw.
    If you want to change the ink color, use this primitive.
12. There is no video in this primitive!
13. 3 is the number for this one!



"Don't fence me in!


7
8
9
1
10
11
2
12
13
3
4
5
6

Chapter 6



You Like Circles, Don't You?

Sam finds some things particularly easy to do just because he is a turtle! From his turtle's point of view, things that might seem difficult to us from above, seem easy for Sam.

When Sam was just a little baby turtle, he only knew how to draw straight lines! One day he was asked about drawing a circle. Now Sam knew what a circle was. He had seen a circle drawn on a piece of paper once, by a very smart old turtle! he decided to think about the problem and try out some ideas as he went along. He knew that if he walked round in a circle, he would eventually have to end up back at his starting position. He also realized that if he started off facing north, he would in the end finish up facing north once again.

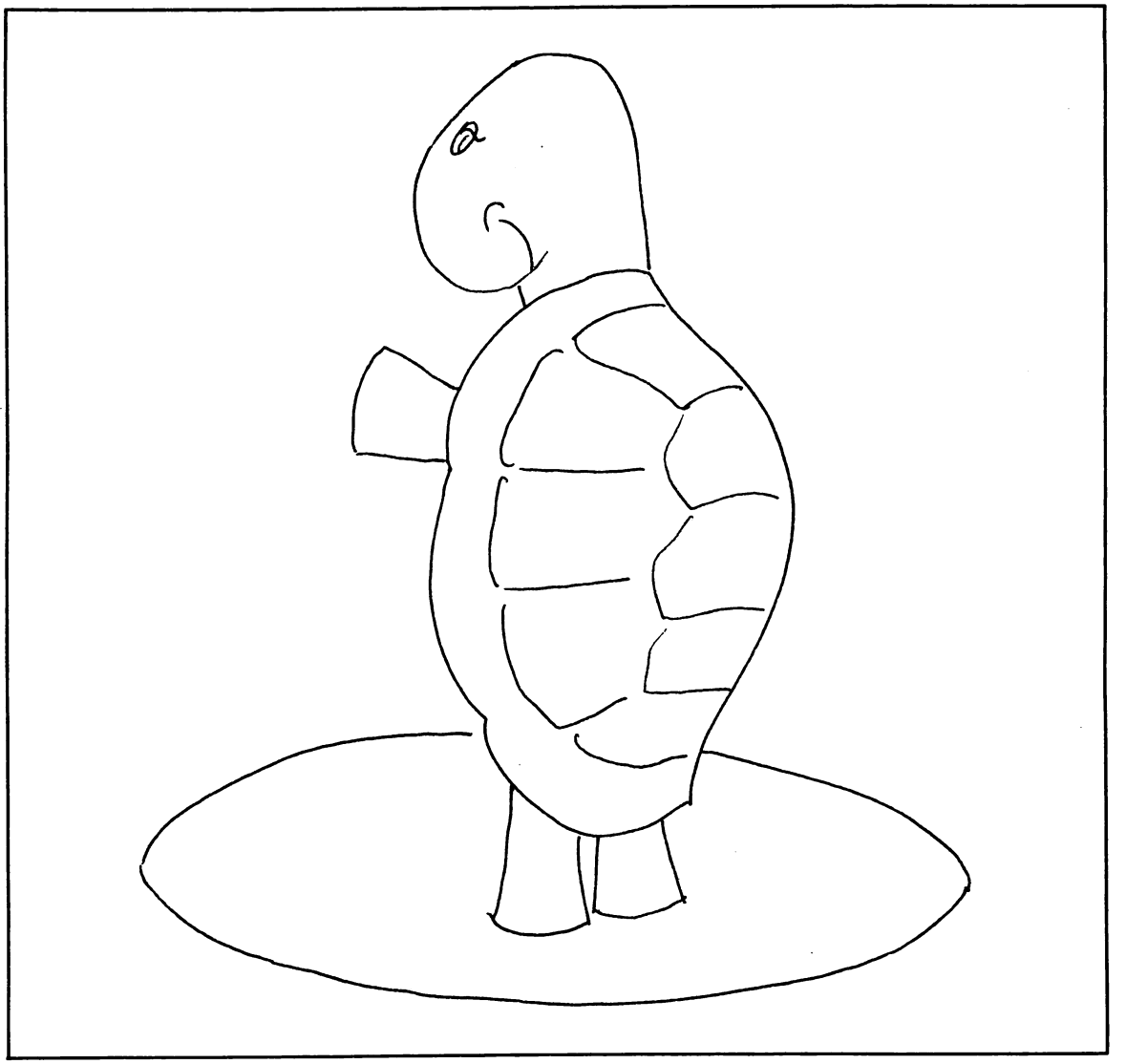
For a while, Sam was a just a little bit stumped. The things that he had tried up to that point hadn't drawn the circle that he had hoped for! Then he asked himself, "what am I really doing when I walk in a circle?" Suddenly it came to him! "What I do is go forward a little bit, and then I turn a little bit; after that I do the same over and over again until I am back at my starting point, facing in my original direction".

Sam tried this out, and sure enough, he ended up back where he started. He also ended up facing in his original direction. He had drawn a circle! Well done Sam!

Sam is a lot like us. He didn't find the answer to his problem right away. He tried some ideas and found that they didn't work. He then tried another idea. Finally, he thought of the idea of *playing turtle*. This seemed to make the problem much easier. Finally, he checked to be sure that he had come in a full circle and that he was facing in the same direction once again. Like Sam, we should always ask "Have I really solved the problem?"

DRAWING CIRCLES BY YOURSELF

Here is the procedure that Sam wrote for



Sam in a circle.

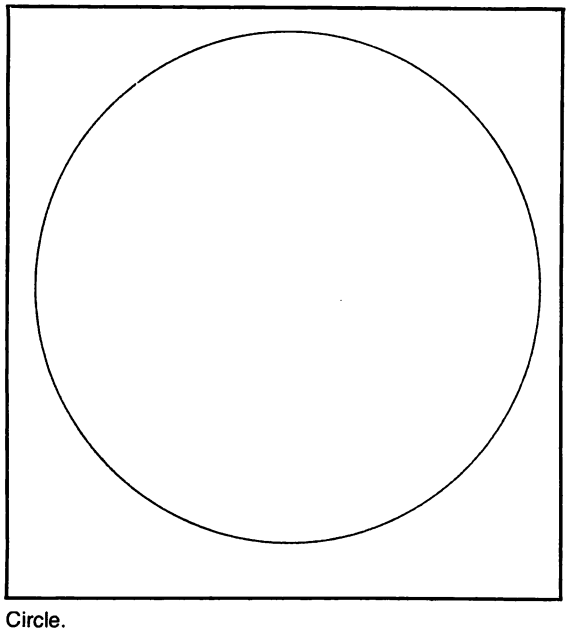
himself. This procedure will draw a circle on your screen.

```
TO CIRCLE
REPEAT 360 [FD 1 RT 1]
END
```

Type in the procedure and then type CIRCLE to see Sam draw the circle on the screen.

Can you see the part that tells Sam to go forward a little bit and then turn right a little bit? That is correct. it is the part in the square brackets that is repeated 360 times. I wonder why it was repeated 360 times? It was repeated 360 times because Sam wanted to get back to where he started from!

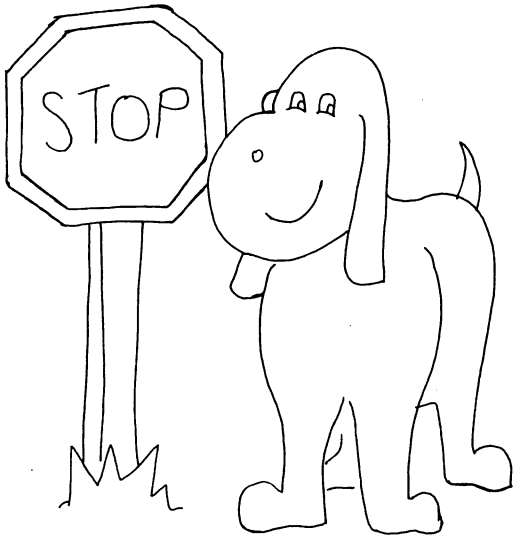
Circles are something that we will want to use a lot. After all, Sam and turtles love drawing circles, just as we do!



Circle.

LETTING SAM DRAW HIS OWN CIRCLES: AUTOMATIC PILOT

The people who "made" Sam decided that it would be nice if Sam knew how to draw circles automatically, all of the time. They wanted him to be able to draw circles without having to think too hard about what he was doing. They decided that they would build the word CIRCLE into Sam's vocabulary, by placing the procedures on the disk that comes with the Logo package.



Sam will understand the words CIRCLER and CIRCLEL if you press the return key just after loading Logo from your disk. (Leave the language disk in the disk drive to do this.) If you do not press the return key at this point, Sam will know nothing at all about circles.

The disk and your Logo files with SQUARE, and BOX, and the other procedures you have saved are part of Sam's memory. As I mentioned before, the thing about a disk is that it is a *permanent* type of memory. When you use only the computer's memory with Sam, you are using a *temporary* memory, which will disappear when you switch Sam off!

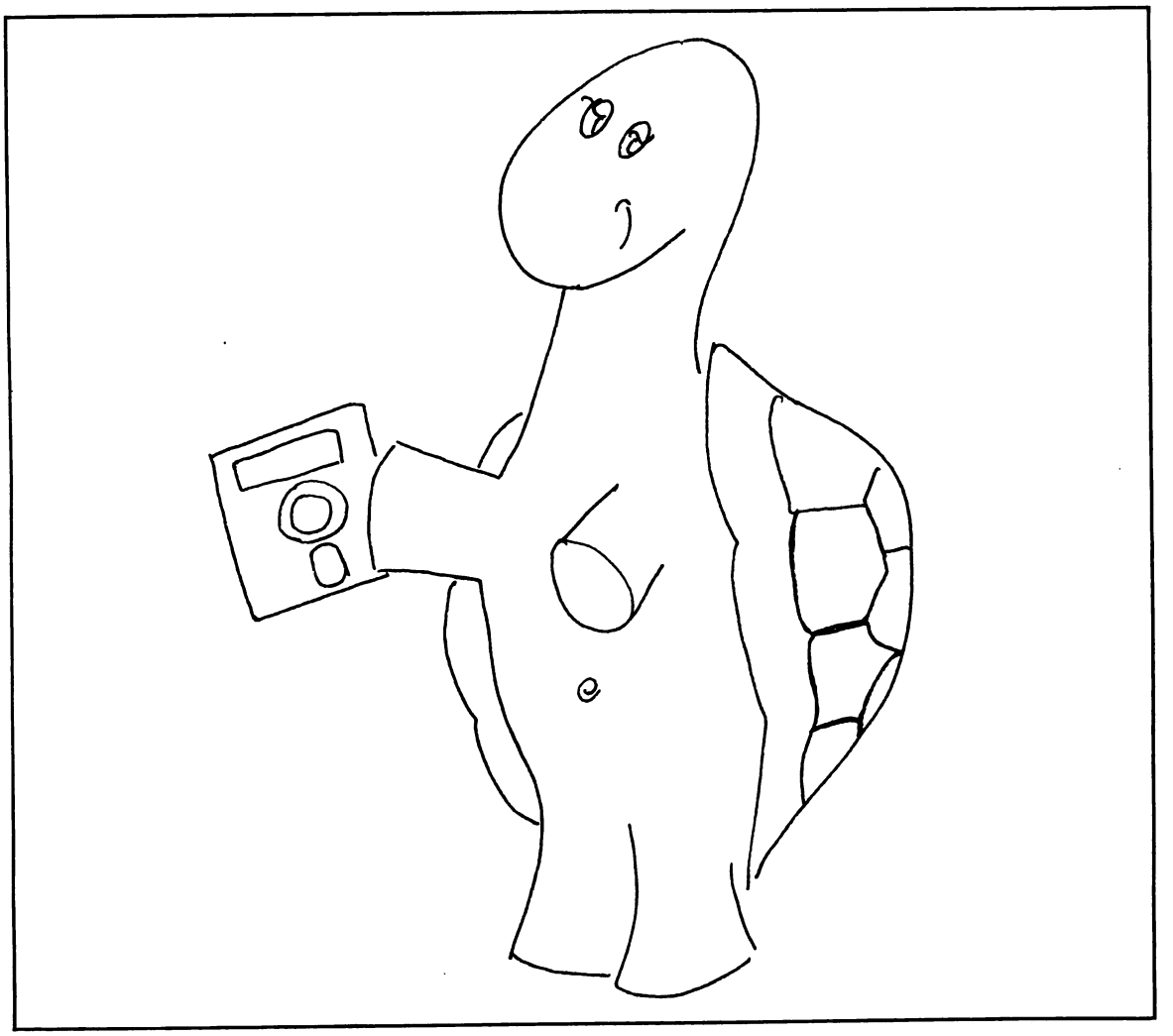
Make sure that you have loaded Sam's extra knowledge from your disk. If you are not sure about this, restart the system, and be sure to press the return key when you see the prompt message displayed on the screen, before you remove the language disk.

When you have done this, you can type in the following to draw a simple circle:

```
CIRCLER 50
```

You should find that Sam drew a circle with a radius of 50 on your screen. The *radius* of a circle is a measure of how large it will be. It is the distance from the center of the circle to the edge of the circle. With Sam, the radius of the circle is measured in turtle steps. Did you notice that he knew when it was time for him to stop? He drew only one complete circle!

Drawing circles this way is much easier than actually having to teach Sam the meaning of the word *circle* each time.



Sam can use a disk to remember things.

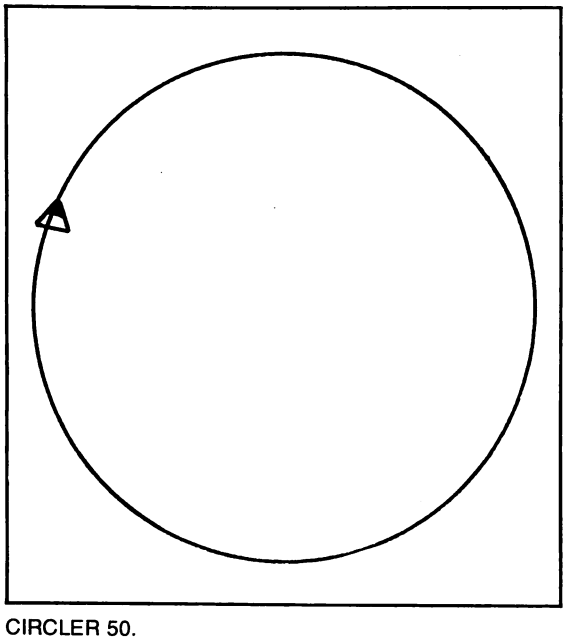
Sam can in fact draw two different kinds of circles. The first is a circle to the right. If you want Sam to circle right, you type CIRCLER followed by the radius. If you want him to draw his other type of circle, a circle to the left, you would type

```
CIRCLEL 40
```

This command will give you a circle to the left with a radius of 40 units. The R in CIRCLER stands for Right and the L in CIRCLEL stands for Left.

Here is a program with circles in it for you to try.

```
TO FOUR.CIRCLE
CIRCLER 50
CIRCLEL 50
FD 40
CIRCLER 20
CIRCLEL 20
END
```



CIRCLER 50.

Type FOUR.CIRCLE to see Sam draw the diagram. Can you see how the procedure did this?

Sam began by drawing two large circles, one to the left and the other to the right. After that, he moved forward a little with the FD 40. Finally, he drew two smaller circles, one to the right and the other to the left.

Here is a challenge for you, can you teach Sam to draw a pair of spectacles using CIRCLER and CIRCLEL?

If Sam can make a diagram with four circles, he should be able to make a better diagram with more circles. I wonder if Sam can use the CIRCLE primitive and the REPEAT command together? Here is a procedure for you to try that will do this.

```
TO SPIN.CIRCLE
REPEAT 6 [CIRCLER 50 RT 60]
END
```

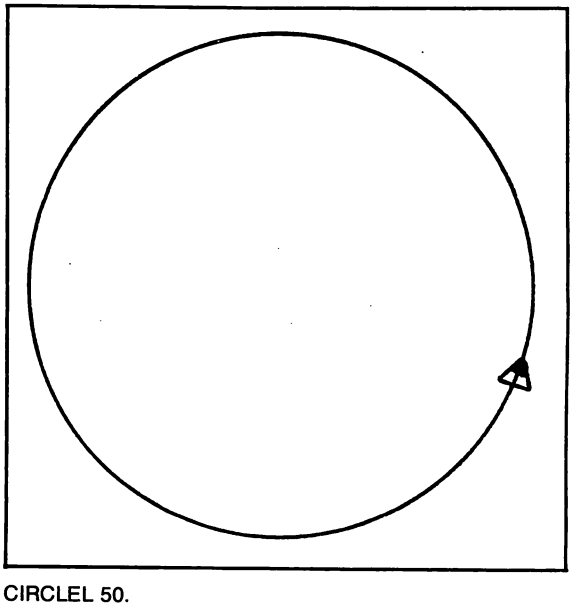
Type SPIN.CIRCLE to see what Sam will do. Can you rewrite the procedure so that you can have 20 circles instead of 6? What about the RT 60 inside the REPEAT command? What will happen if we make changes to it?

Now, let me see, when I pressed the return key and loaded the CIRCLER and CIRCLEL procedures into Sam's memory, the disk drive started and stopped, then and ran for quite a long time! Did you notice this when you pressed the return key? I wonder if anything else was loaded into Sam's memory?

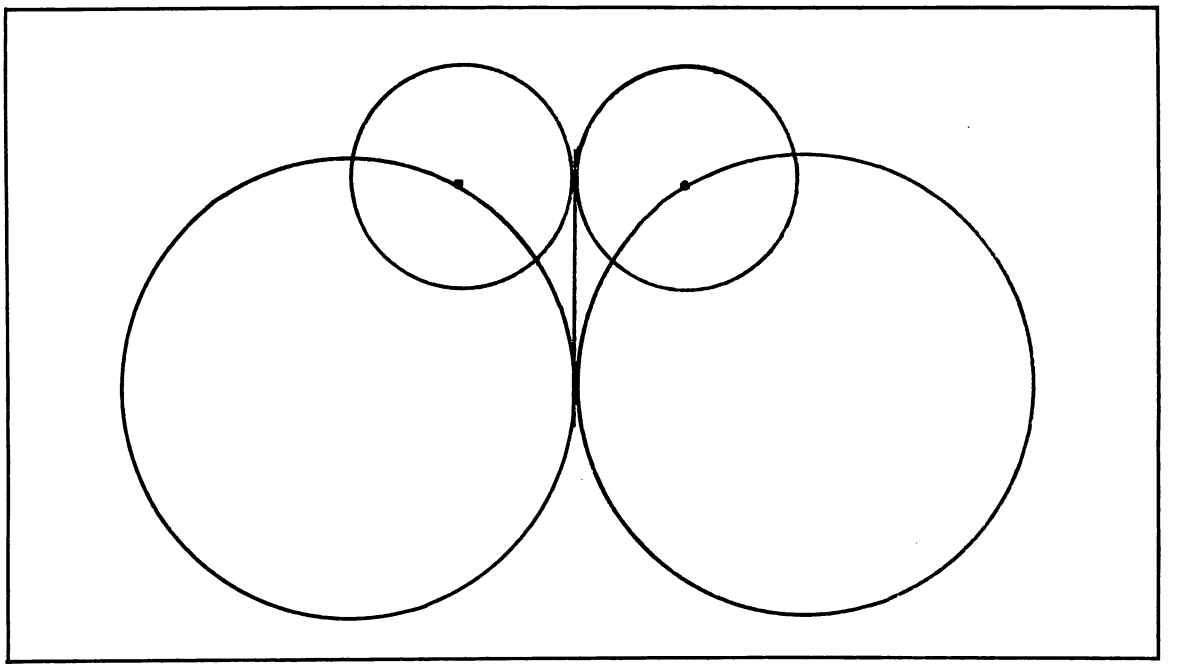
HOW USEFUL IS PART OF A CIRCLE?

I was right! There was something else loaded into Sam's memory! Sam also knows how to draw parts of circles. Actually, a part of a circle is very useful. It is so useful that it is given a name of its own. Part of a circle is called an *arc*. The illustration on page 109 shows a number of arcs.

You will find that you can use arcs to draw a lot of things. The illustration on page 110 shows just a few of the more obvious things that you can draw with them.



CIRCLEL 50.



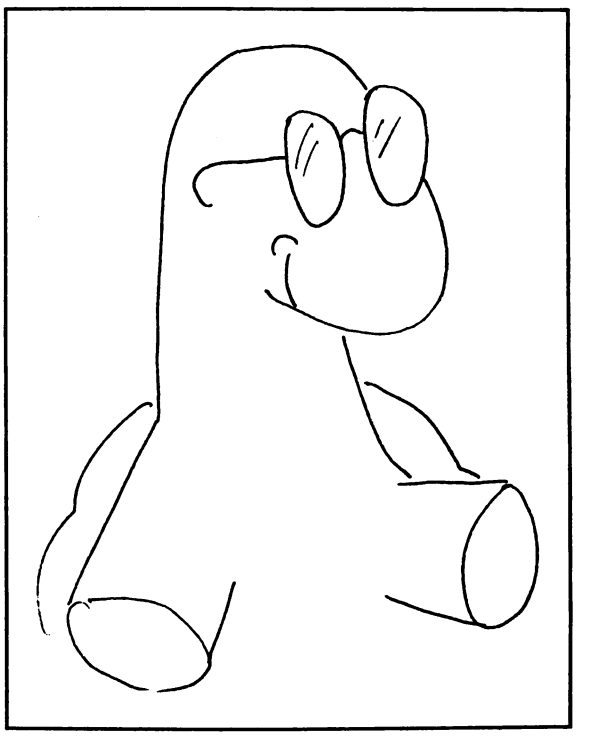
FOUR.CIRCLE.

If you want to tell Sam to draw an arc, type the following:

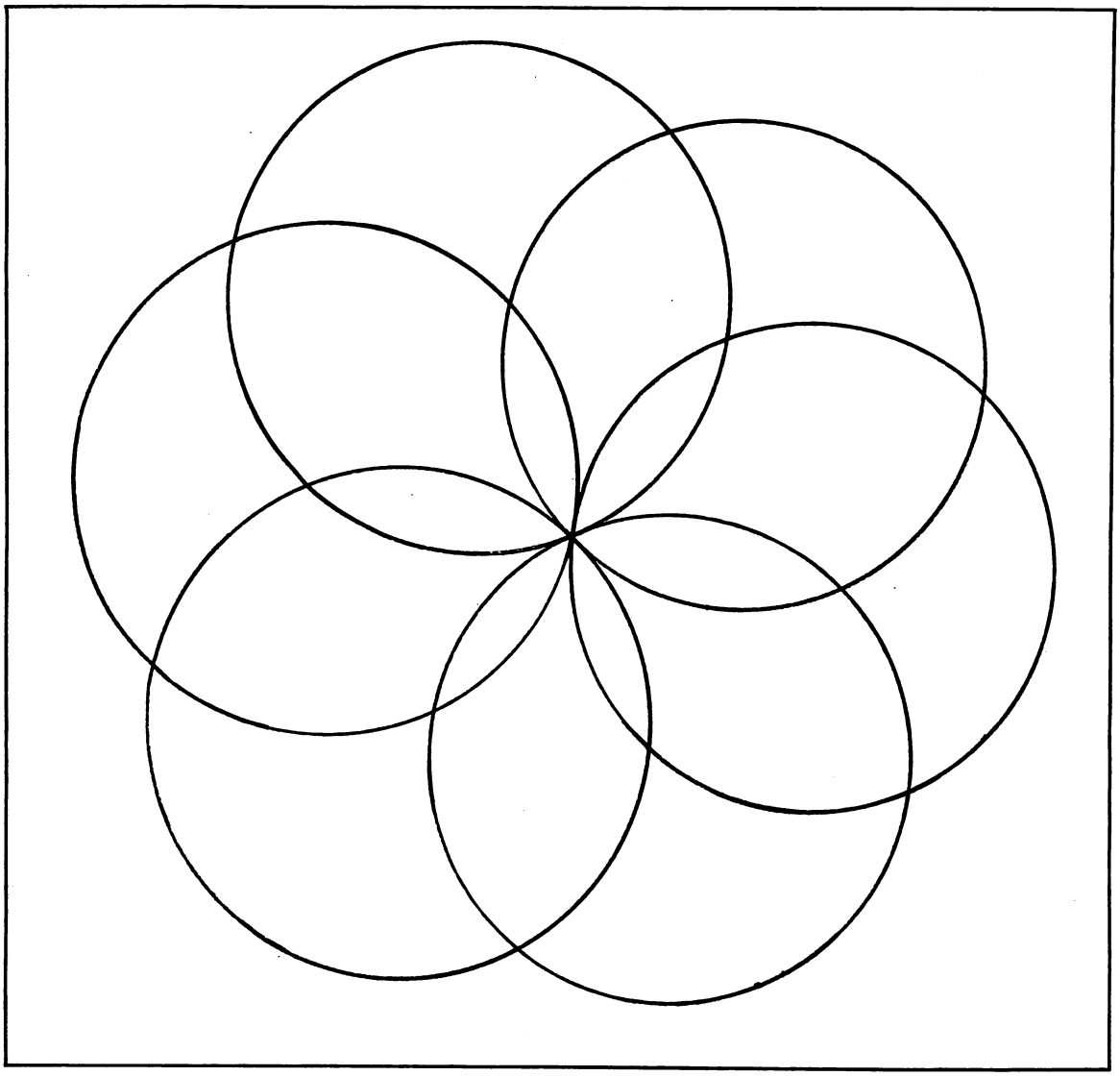
```
ARCR 30 150
```

This will give you an arc of radius 30, and it will give you 150 *degrees of arc*. The number at the end of the command will determine how long your arc will be. Can you discover how many degrees of arc there are in a circle? That is, what would you have to type instead of the 150 to make Sam draw an arc that will turn out to be a complete circle?

Did you find that there are 360 degrees of arc in a circle? There is that number 360 again! As I said before, 360 is a very special number for Sam! This is because, Sam lives in a world of geometry, and in geometry, 360 is a number that occurs over and over again. You have looked at these ideas before, but they are such a great help when you are writing procedures that they are worth looking at once more.



Can you draw a pair of glasses for Sam?



SPIN.CIRCLE.

The number 360 appears quite a lot. Perhaps you have noticed that some other numbers have appeared over and over again. For example, 120, 60, 30, and 90 are very common numbers in Logo and in geometry. So what is special about these numbers? The numbers are special because they can be divided evenly into 360.

Did you notice that the command

```
REPEAT 4 [FD 100 RT 90]
```

has a 4 and a 90 in it. 4 times 90 is, of course, 360!! (If you recall, the procedure drew a square!).

Do you remember the first procedure that we used to draw a triangle?

```
TO TRIANGLE
```

```
REPEAT 3 [FD 100 RT 120]
END
```

Guess what, there is a REPEAT 3, and a RT 120. 3 times 120 is, of course, 360!!

So much for 360 and other special numbers, let's look at the arcs again!

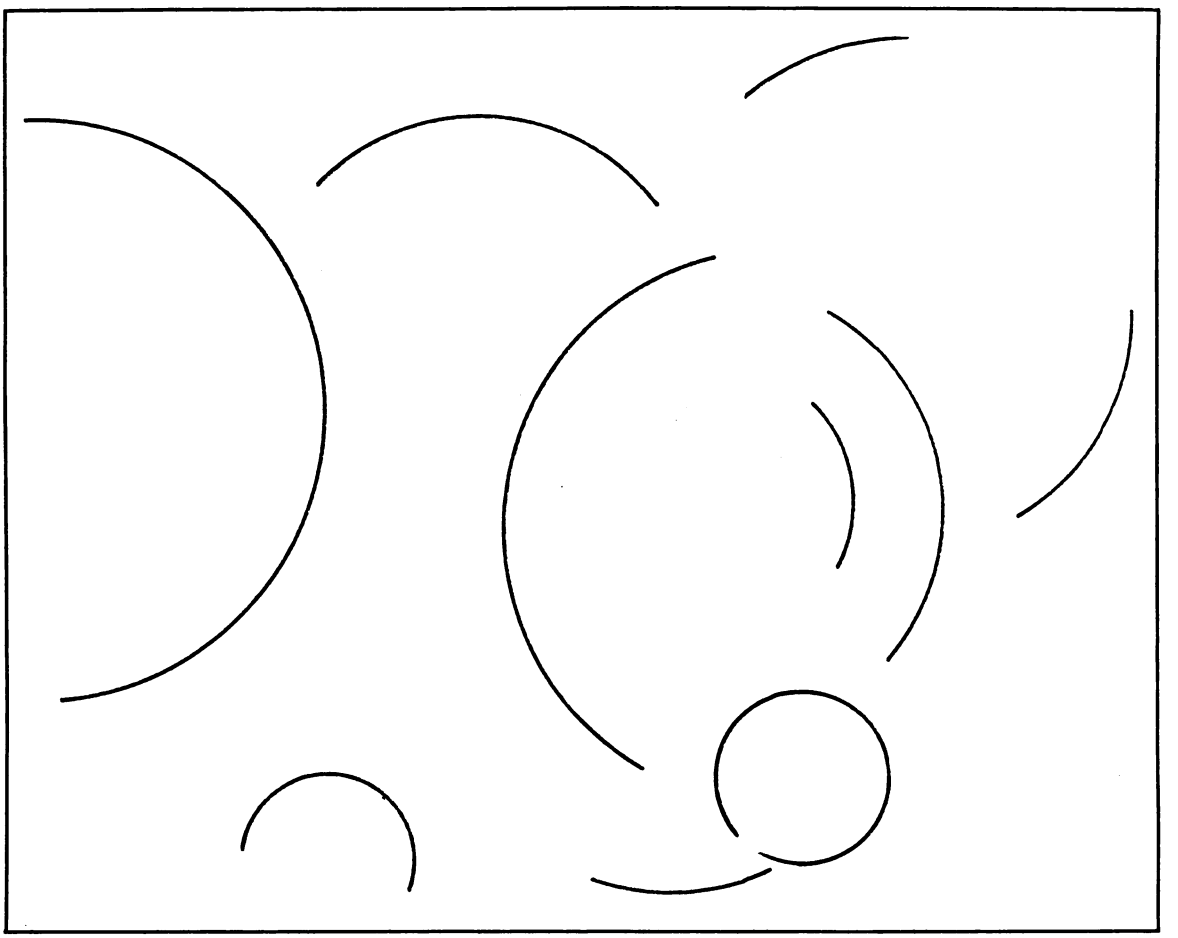
Sam can ARCL as well as ARCR, in the same way that he can CIRCLEL as well as CIRCLER. Try out a few ARCRs and ARCLs to become familiar with the numbers that they use and what they can do.

When you are familiar with circles and arcs, try the following procedure.

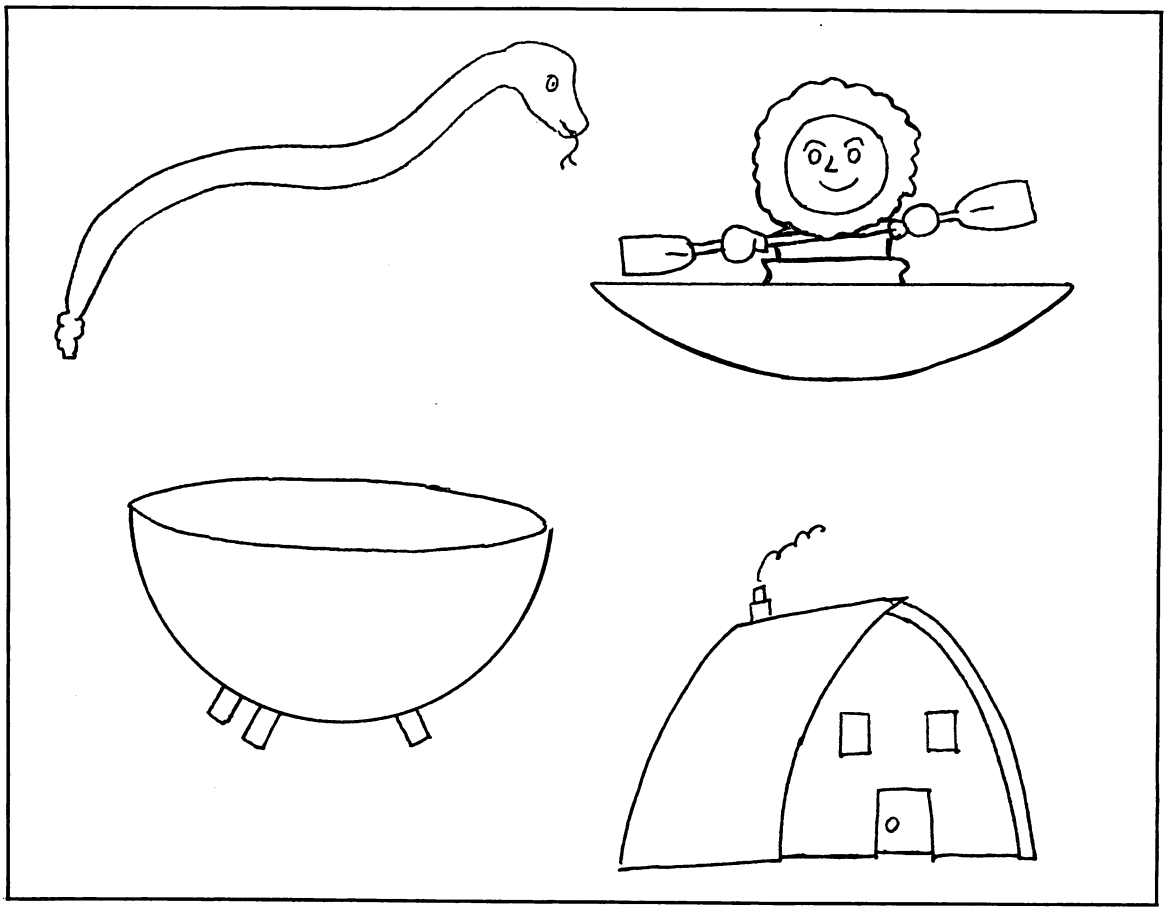
```
TO FLOWER
REPEAT 9 [PETAL]
END

TO PETAL
ARCR 50 90
RT 100
ARCR 50 90
END
```

To see what this procedure will do, type FLOWER. Can you rewrite the FLOWER procedure to add a curved stalk to the flower?

Here is a possible solution. Lets look at it in English statements first of all.



Arcs.



Things made from arcs.

To draw a better flower
Draw the stalk
Draw the flower
End

Stalk
Draw an arc
End

Flower
Draw the petals
End

Here is a procedure that was written from the English list.

```
TO BETTER.FLOWER
STALK
FLOWER
END

TO STALK
PU
BK 100
PD
ARCR 500 20
END

TO FLOWER
REPEAT 9 [PETAL]
END
```

```
TO PETAL
ARCR 30 90
RT 100
ARCR 30 90
END
```

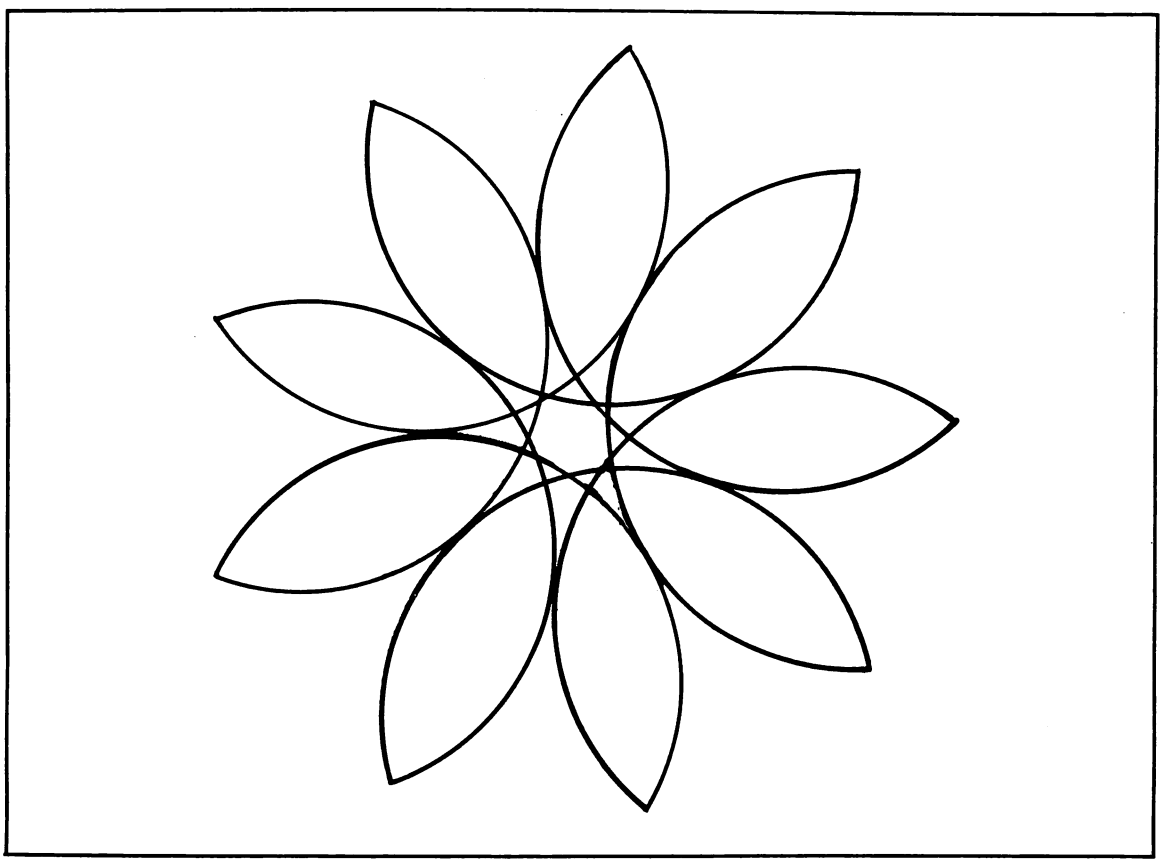
The procedures FLOWER and PETAL are almost the same as before! If they are already in the computer, you will not need to retype them. You can use the editor to change the values in the procedures! Can you tell why the values were changed? Correct, to get our new flower to fit on the screen! To see your new flower, type BETTER.FLOWER and then press the RETURN key.

Can you write a procedure that will draw a very small flower on the screen? When you have done that, can you write a procedure that will draw a group of small flowers on the screen?

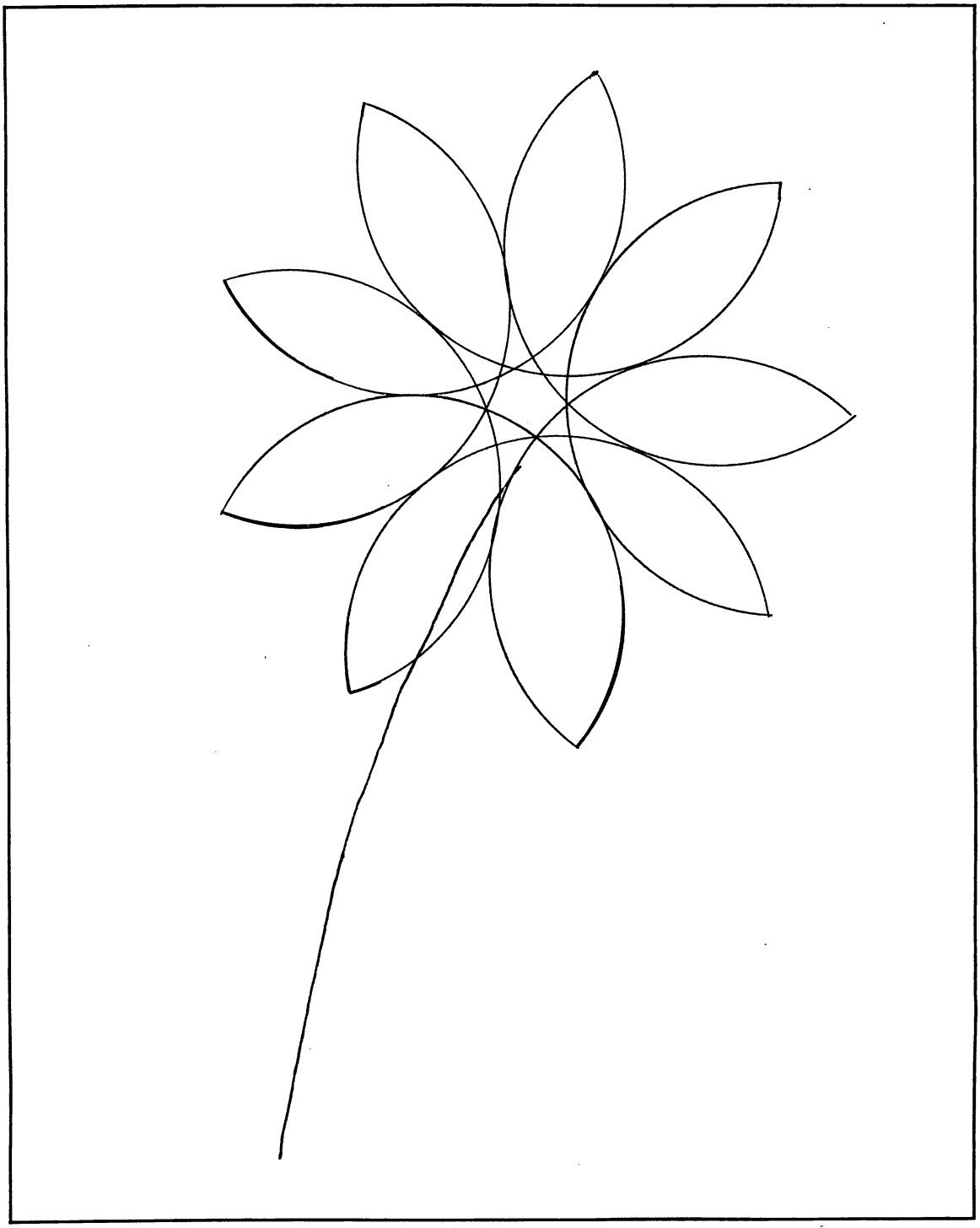
I wonder what other patterns we can draw with arcs?

Wiggle Waggle with Sam

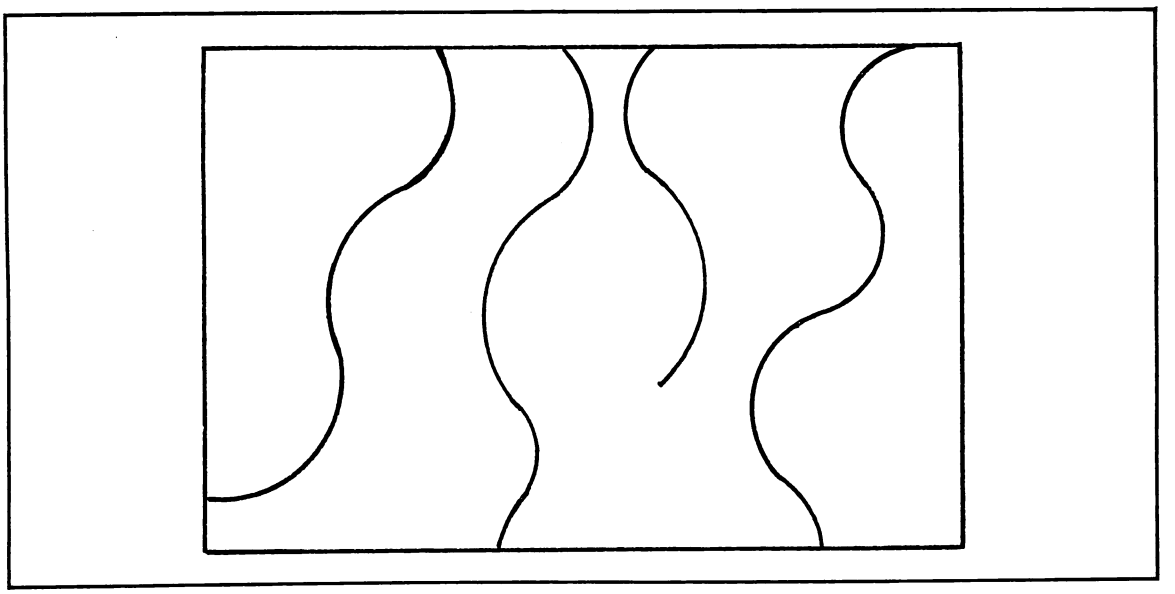
Let's see if we can teach Sam how to wiggle waggle across the screen. Wiggles are really a string of arclefts and arcrights strung together. Perhaps we could use the REPEAT primitive to string them together. Of course, turtles don't like snakes very much and a wiggle looks a bit like a snake, but I think that Sam will draw the diagram for us anyway! We will



FLOWER.



BETTER.FLOWER.



WIGGLE.WAGGLE.

have to ask him to draw the wiggles properly of course! Try typing in the following procedure to see Sam wiggle.

```
TO WIGGLE.WAGGLE
RT 60
REPEAT 100 [ARCL 40 90 ARCR
40 90]
END
```

Type WIGGLE.WAGGLE to see Sam do his stuff.

Can you say why we had to tell Sam to RT 60 just before the REPEAT primitive? What would happen if we used RT 30 or LT 80 instead? Perhaps it might be interesting to try some different primitives and values here! How about changing the RT 60 to RT 45!

Lots of Arcs, Lots of Pictures

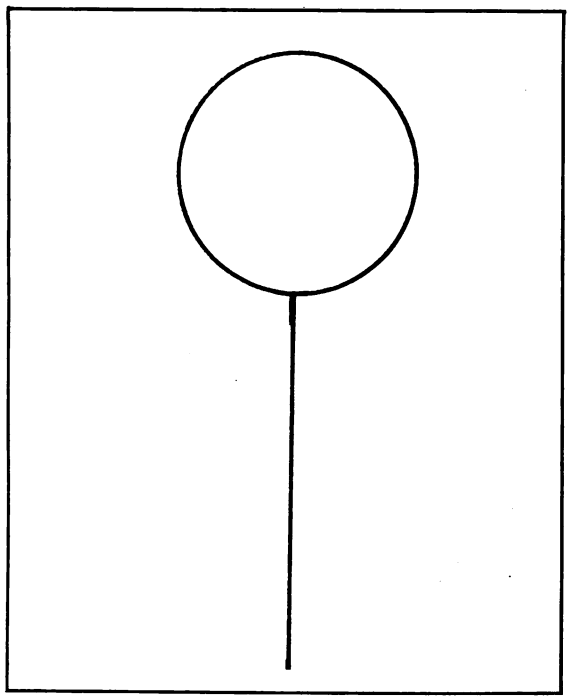
Suppose that we wanted to teach Sam the meaning of the word lollipop! I should think that Sam likes lollipops quite a lot. Let me see . . . a lollipop is simply a circle with a stick on it, or is it a stick with a circle on it? Let's think of it as a stick with a circle on it. We should teach Sam to draw the stick first and then the top of the lollipop.

This means that our first procedure should look like this.

```
TO LOLLIPOP
STICK
POP
END
```

We used two subprocedures, STICK and POP. The POP procedure will draw the most interesting part of the lollipop, the candy part! Here are the two missing procedures.

```
TO STICK
FD 60
END

TO POP
LT 90
CIRCLER 20
END
```



LOLLIPOP.

Why is there a LT 90 in the POP procedure? What would happen if you left the LT 90 out, or if you changed the value 90 in some way?

If you want to see Sam draw his lollipop, you would have to type the following into the machine.

```
TO LOLLIPOP
STICK
POP
END

TO STICK
FD 60
END

TO POP
LT 90
CIRCLER 20
END
```

Type LOLLIPOP to see Sam draw the lollipop.

We didn't really have to use subprocedures to draw the lollipop, but it is a very good idea to use a lot of subprocedures if we can. Start thinking in terms of subprocedures now!

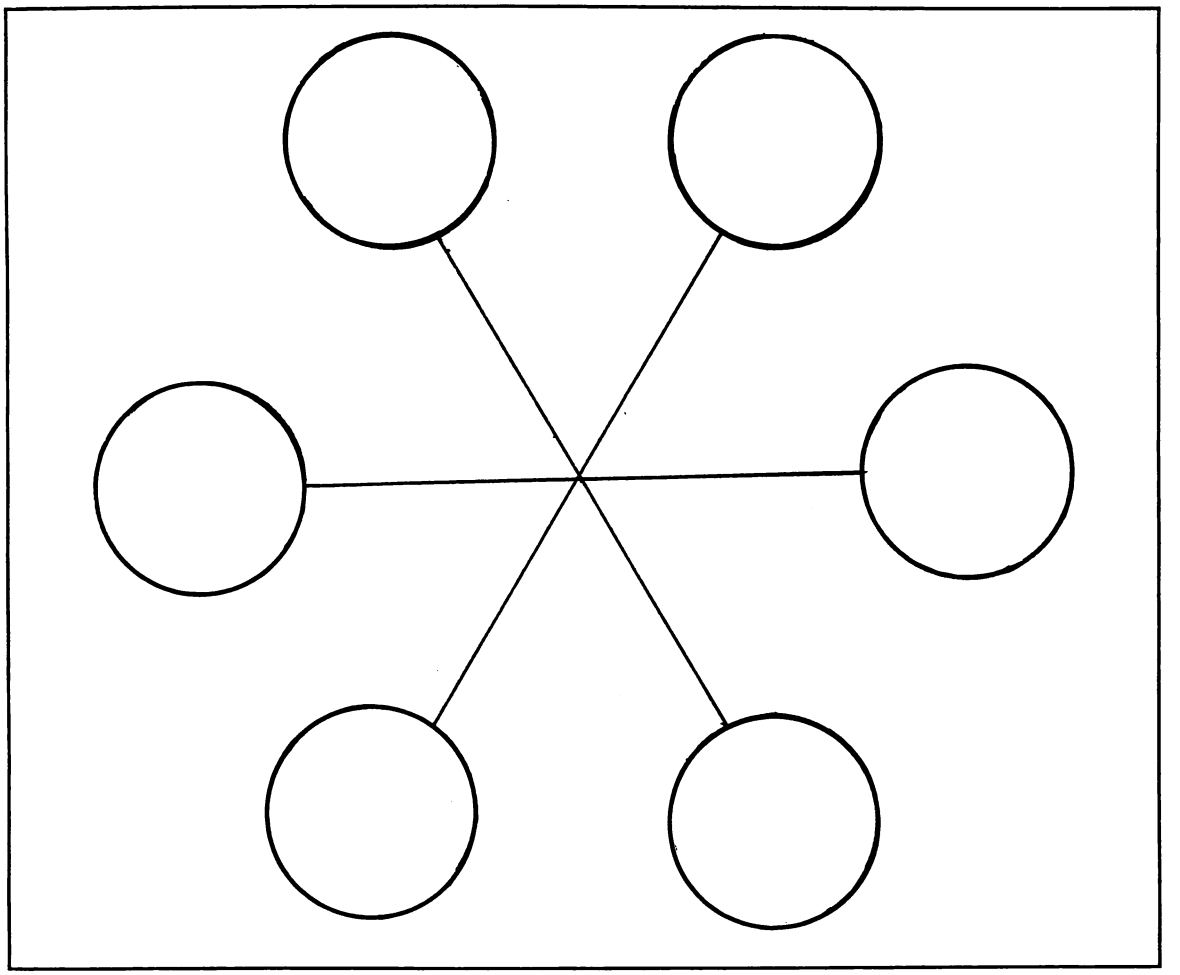
I wonder if we can teach Sam to do other things with the LOLLIPOP procedure now that he knows what a lollipop is? What could be nicer than one lollipop? I know! A bunch of lollipops. We could draw them all in a circle! Why not try this one for yourself, and then look at the next procedure to see one way that it can be done.

```
TO LOLLIPOP.BUNCH
REPEAT 6 [LOLLIPOP MOVE]
END

TO MOVE
RT 90
BK 60
RT 60
END
```

To see Sam draw the bunch of lollipops, type LOLLIPOP.BUNCH.

What do you think the procedure MOVE does? Correct! it will move Sam to his next starting point so that he can start to draw his next lollipop. There is a 360 in our procedure somewhere; can you see where it is? Yes, it is the 6 times 60! The 6 comes from the REPEAT procedure, and the 60 comes from the RT 60 that will get Sam ready for the next lollipop. If you got an I DON'T KNOW HOW TO LOLLIPOP message from Sam, then you will need to reload the procedure LOLLIPOP from disk.

I wonder what other things we can create with circles? Perhaps we could get Sam to draw an umbrella for us by using circles and arcs! Let's decide what sort of umbrella we want. A picture of the umbrella we will try to draw is on page 116.



LOLLIPOP.BUNCH.

We would write the following English list to describe how we would draw the umbrella.

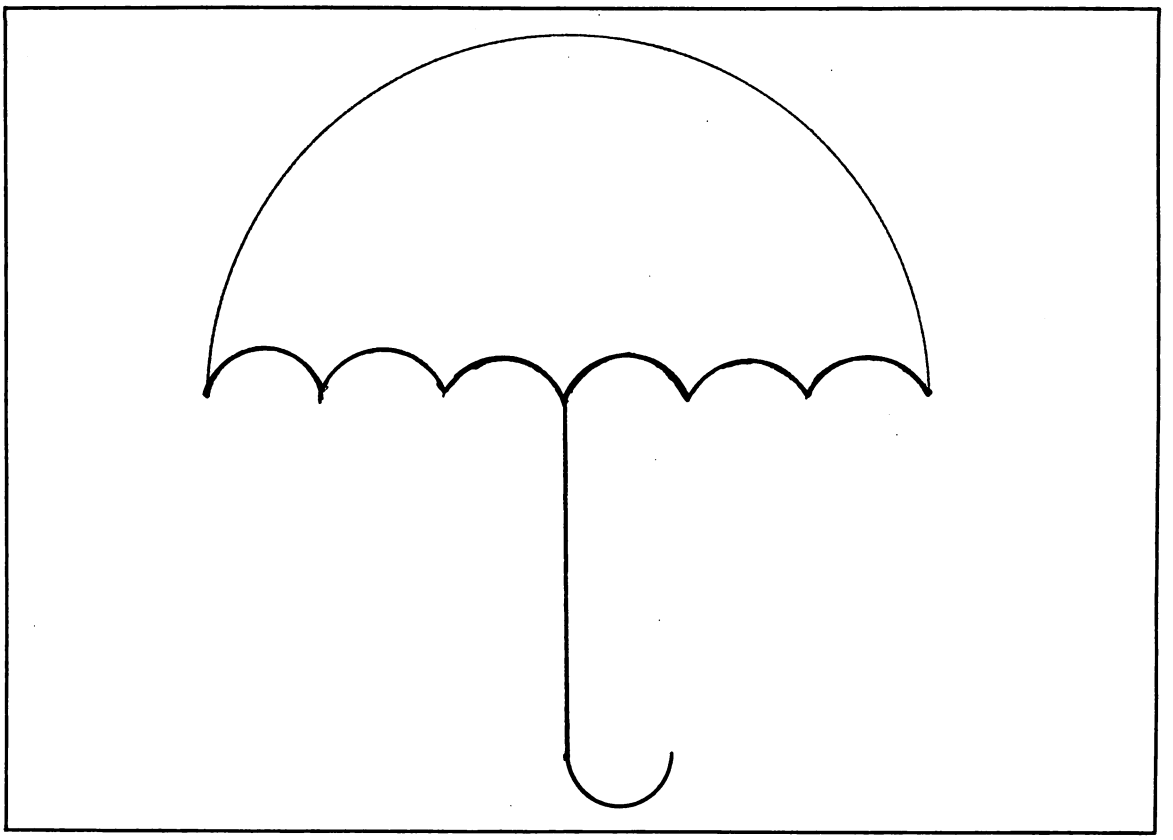
Umbrella
Handle
Top
End

Unfortunately, this is not detailed enough, lets break the list into smaller pieces like this.

Handle
Crook
Stem
End

Top
Wavy part
Dome part
Wavy part
End

Now that we have an idea of the English listing, let's try it in Logo. Here is what you would have to type in. When the procedure was



Umbrella.

actually written though, each little part was written and tested by itself. For instance, the TOP procedure was typed into the machine first of all and then executed by typing TOP to make sure that the top of the umbrella looked just right.

```
TO UMBRELLA
HANDLE
TOP
END

TO HANDLE
RT 180
ARCR 10 180
FD 60
END

TO TOP
REPEAT 3 [ARCL 10 180 RT 180]
ARCR 60 180
RT 180
REPEAT 3 [ARCL 10 180 RT 180]
END
```

To see the umbrella, type UMBRELLA. This is one of the most complicated procedures so far! Can you see how all of the numbers are related to each other in the procedure, there are a lot of 180's and 90's for instance. How did we know that we needed a 60 in ARCR 180? Try changing the procedure a little at a time to see what will happen!

If you wanted to, you could break the procedure TOP into subprocedures. Here is

what the new TOP procedure would look like.

```
TO TOP
WAVY.PART
DOME.PART
WAVY.PART
END

TO WAVY.PART
REPEAT 3 [ARCL 10 180 RT 180]
END

TO DOME.PART
ARCR 60 180
RT 180
END
```

This would make your complete procedure look like this!

```
TO UMBRELLA2
HANDLE
TOP
END

TO HANDLE
RT 180
ARCR 10 180
FD 60
END

TO TOP
WAVY.PART
DOME.PART
WAVY.PART
END

TO WAVY.PART
REPEAT 3 [ARCL 10 180 RT 180]
END

TO DOME.PART
ARCR 60 180
RT 180
END
```

Which procedure do you prefer? Both are pretty good! However, I think the second one is more sophisticated and it is most like the English listing! You can change the HANDLE procedure in the same way if you like. Can you write the subprocedures that you would need to do that?

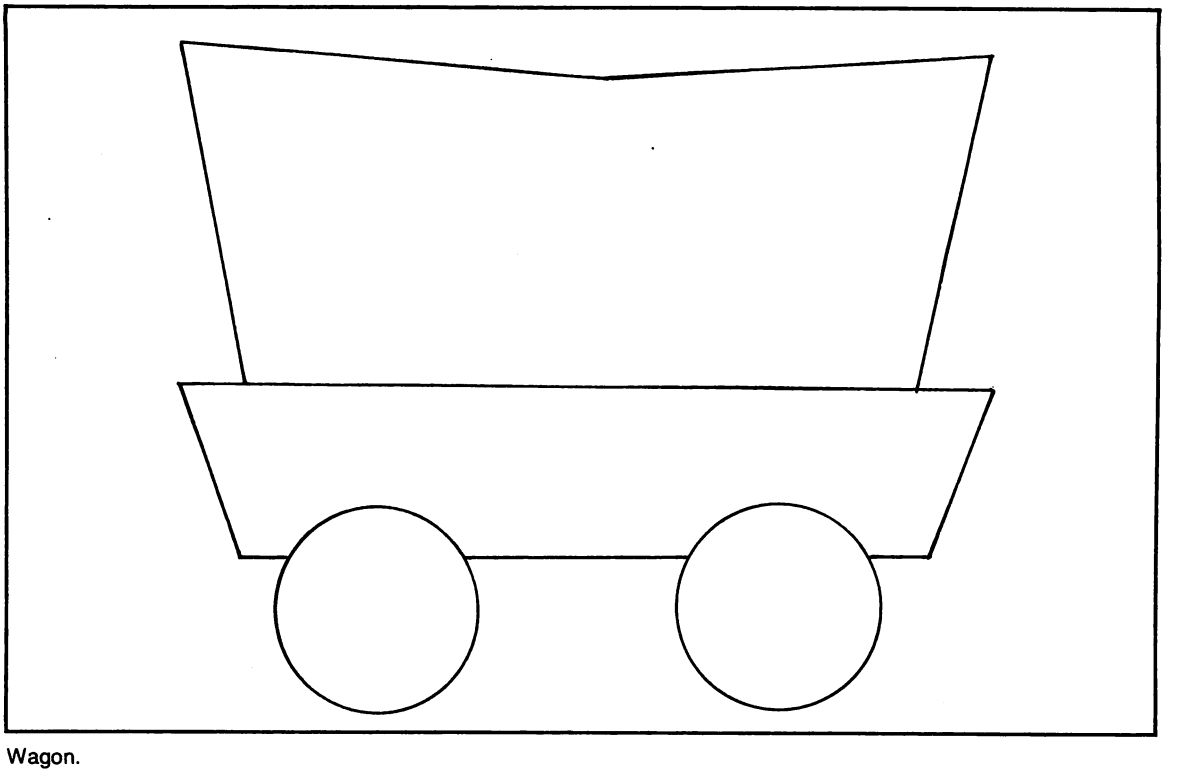
SUMMARY

Here is a summary of the things that are new in Chapter 6. Following the summary are some challenges for you, a quiz, and a crossword puzzle.

1. You can teach Sam to draw a circle by telling him to go forward a little and then right a little until he returns to his original position.
2. Sam has already been taught the meaning of the word circle in Apple Logo.
3. Sam knows how to CIRCLER and CIRCLEL.
4. CIRCLER 50 will draw a circle to the right with a radius of 50 steps.
5. CIRCLEL 50 will draw a circle to the left with a radius of 50 steps.
6. Sam also knows how to draw arcs. He can draw arcs to the left or to the right.
7. ARCR 50 150 will draw an arc to the right with a radius of 50 and 150 degrees of arc. ARCL 50 150 will draw an arc to the left with a radius of 50 and 150 degrees of arc.
8. You can use these commands to draw designs and pictures.

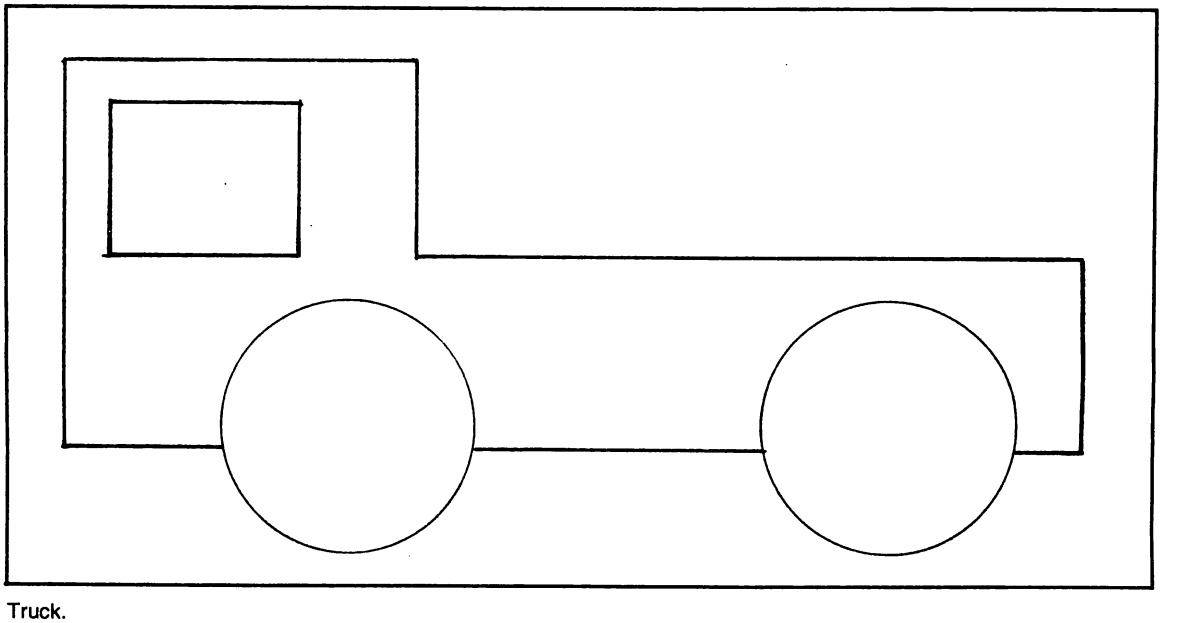
CHALLENGES

1. Draw a wagon with wheels.



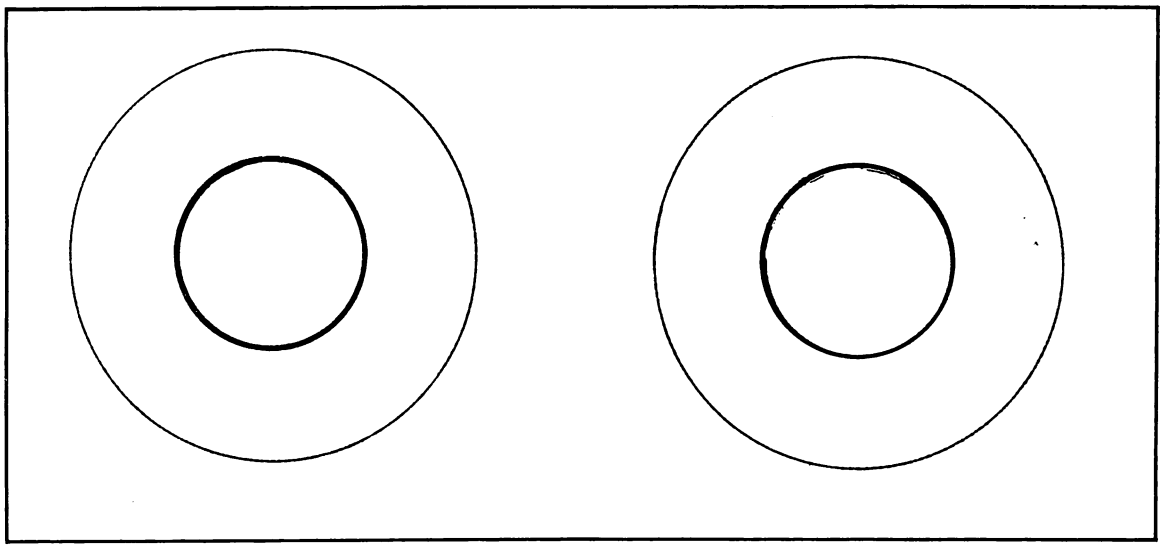
Wagon.

2. Draw a truck with wheels.



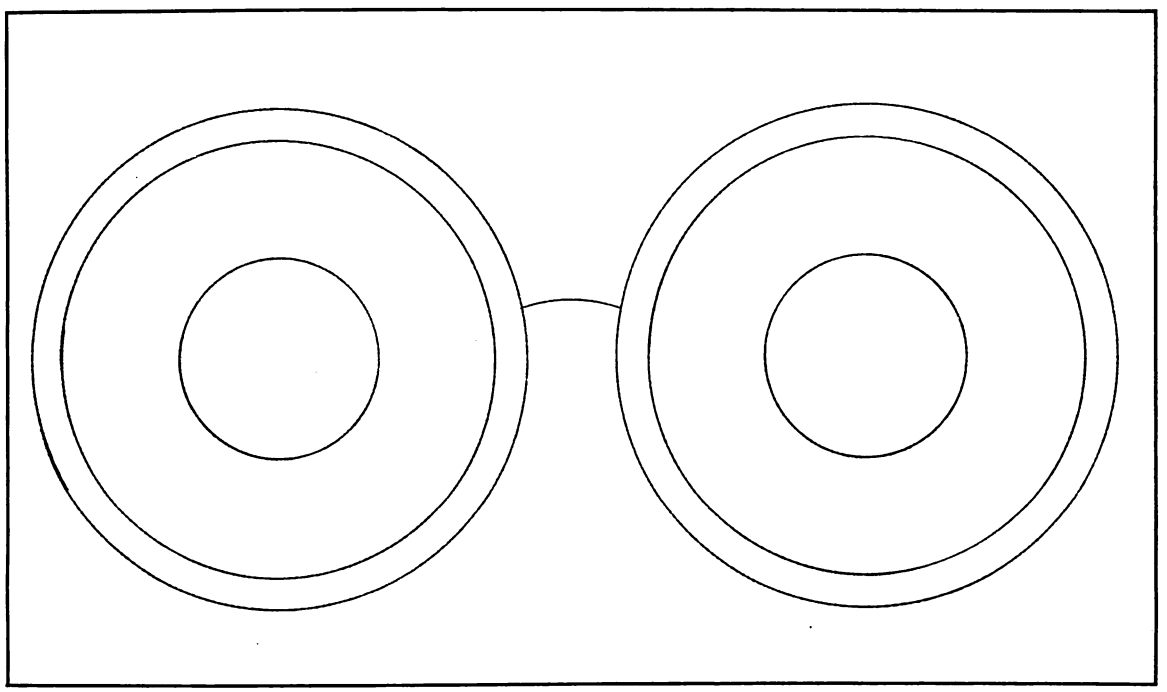
Truck.

3. Draw a pair of eyes.



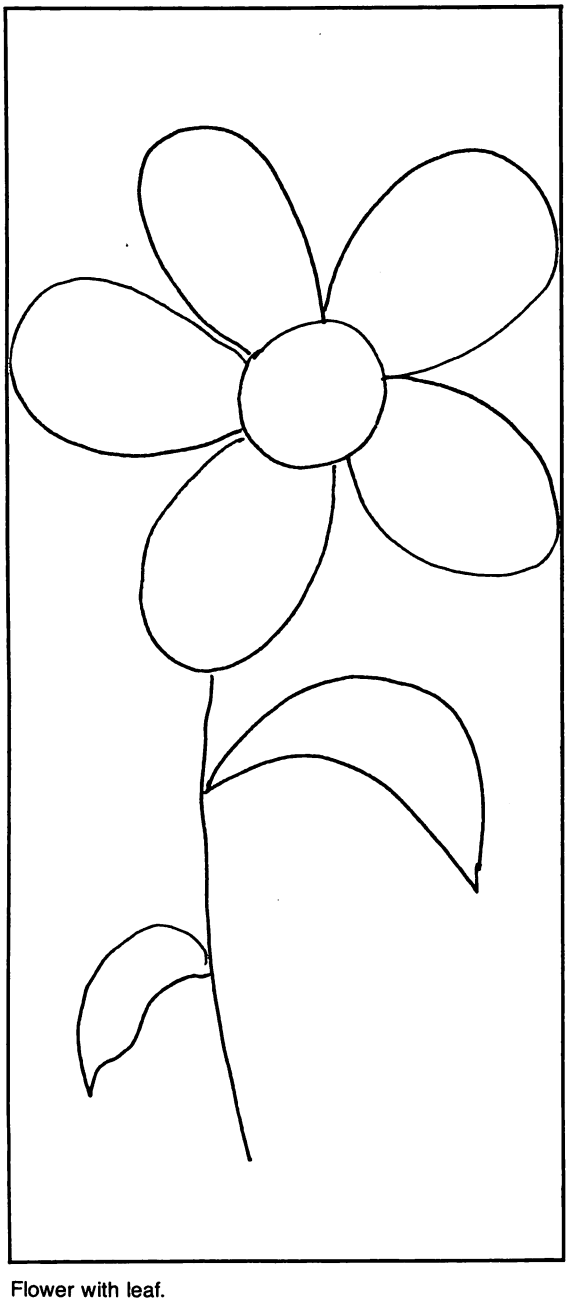
Eyes.

4. Draw a pair of spectacles on the eyes.



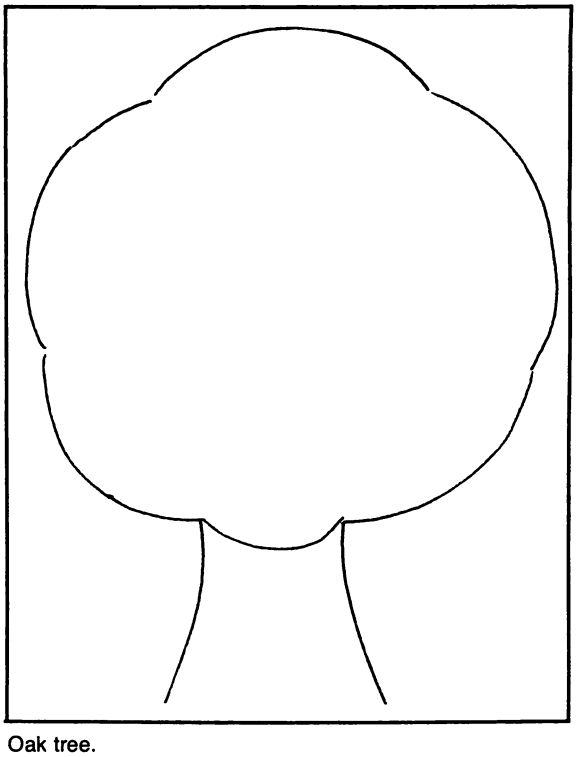
Spectacles and eyes.

5. Draw a picture of a flower with a leaf on it.



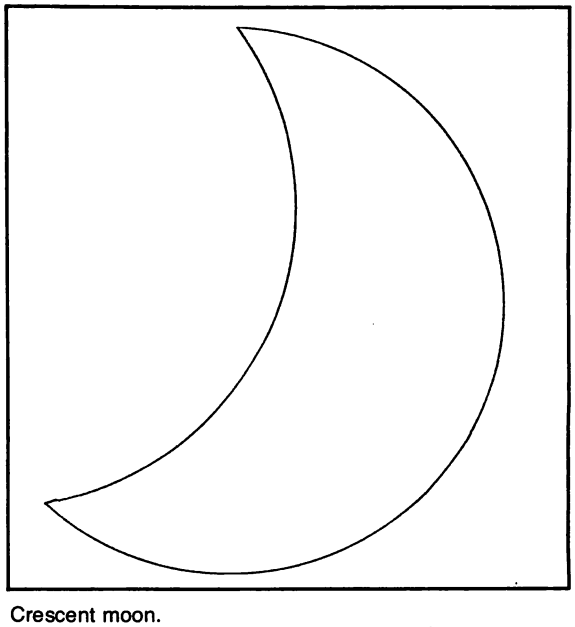
Flower with leaf.

6. Draw a picture of a tree that looks like this.



Oak tree.

7. Draw a crescent moon.



Crescent moon.

QUIZ

1. You can teach Sam to draw a _____ by telling him to go forward a little and then right a little until he returns to his original position.
2. Sam has already been taught the meaning of the word _____ in Apple Logo.
3. Sam knows how to CIRCLER and _____.
4. CIRCLER 50 will draw a circle to the _____ with a radius of _____ units.
5. CIRCLEL 50 will draw a circle to the _____ with a radius of 50 steps.
6. Sam also knows how to draw arcs. He can draw arcs to the _____ or right.
7. ARCR 50 150 will draw an arc to the _____ with a radius of 50 and 150 degrees of arc. ARCL 50 150 will draw an arc to the left with a radius of 50 and 150 _____.
8. You can use these commands to _____.

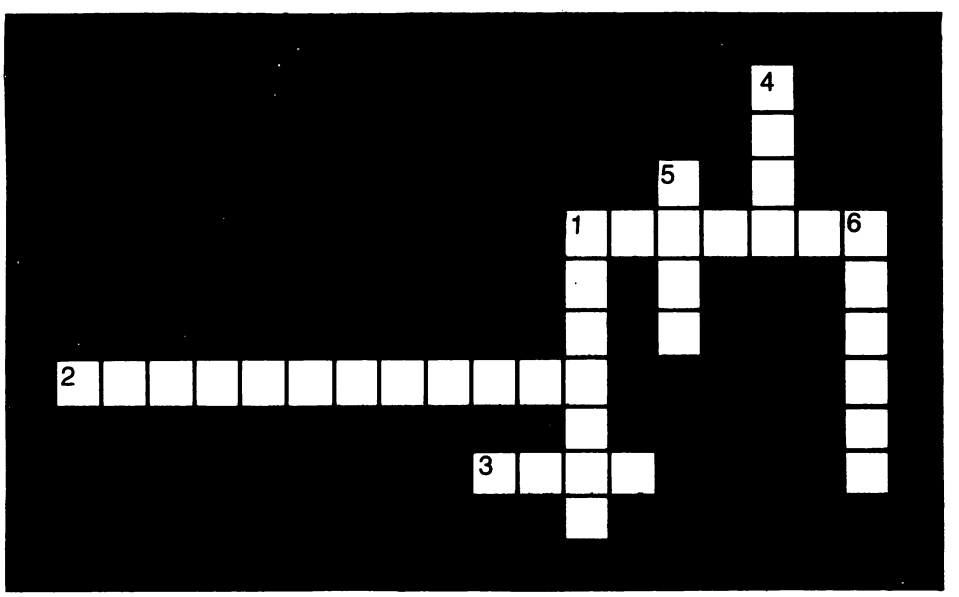
CROSSWORD PUZZLE

Clues Across

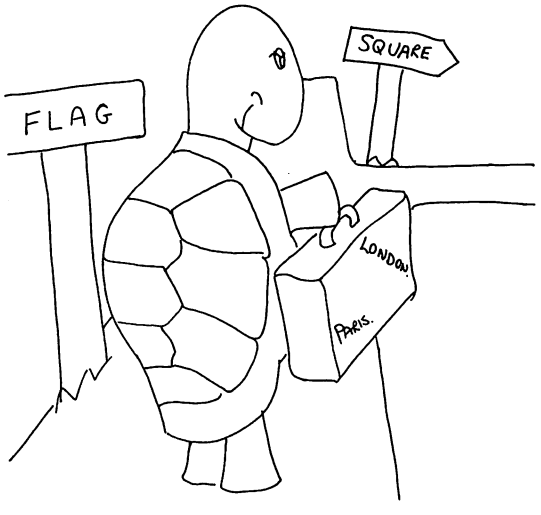
1. This primitive tells Sam to draw a circle to the right.
2. This is what the number 45 in ARCL 30 45 represents.
3. Sam's permanent memory.

Clues Down

4. This will tell Sam to draw an arc to the left.
5. This will tell Sam to draw an arc to the right.
1. Draw a circle to the left Sam!
6. This is what the number 50 in CIRCLER 50 represents.



Chapter 7



Telling Sam to Draw Large and Small

Sam is an amazing fellow. He can actually understand the idea of large and small; that is, you can tell him if you want him to draw a large diagram or a small diagram. This is done by sending him a message, a number usually, that will tell him how large or how small the picture is to be.

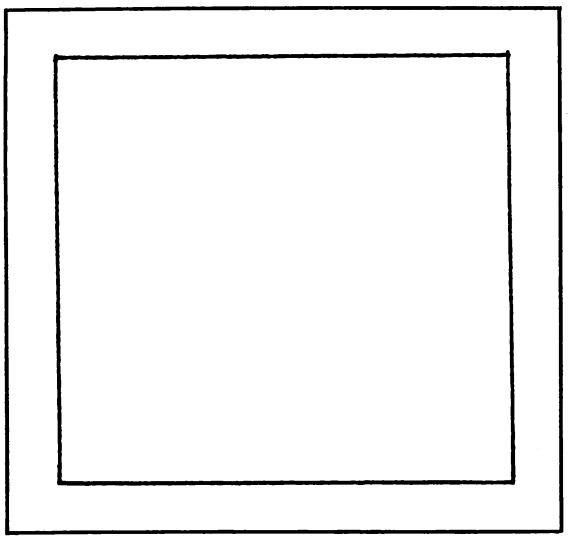
The number that you send him is referred to as an *input*. You often send him this number when you ask him to carry out a procedure. For instance, instead of telling Sam to simply SQUARE, you might tell him to SQUARE 50 or to SQUARE 120.

The first statement would produce a square with sides of 50 steps each; the second would produce a square with sides of 150 steps each. Of course, we would have to teach Sam how to take care of these extra numbers. He certainly will not be able to cope with the extra numbers without your help. Here is what you could type to get Sam to take care of those extra numbers in your SQUARE procedure.

```
TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END
```

When you have typed in this procedure, type the command **SQUARE 100** to see what happens. You should find that you get a square with sides of 100 units each. Now, try typing SQUARE 50. What did you get this time? This time, the procedure should have drawn a square with sides of 50 units each! Experiment with the procedure by using different number values for the side of the square.

Let's look at the procedure a little more closely. The TO SQUARE part is the same as before, but the :SIZE is new. We can say :SIZE in two ways in Logo: we can say "dots SIZE" or we can say "the value in the *variable* SIZE". The second one is more descriptive, but the

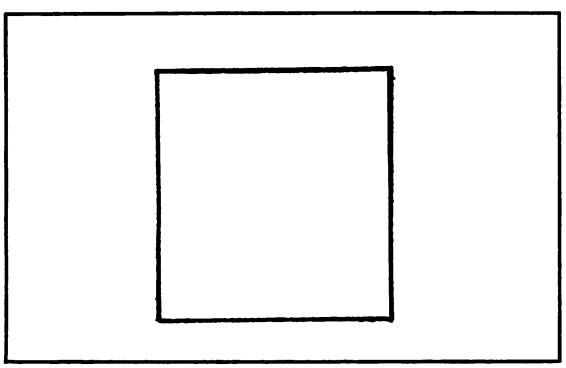


Square 100.

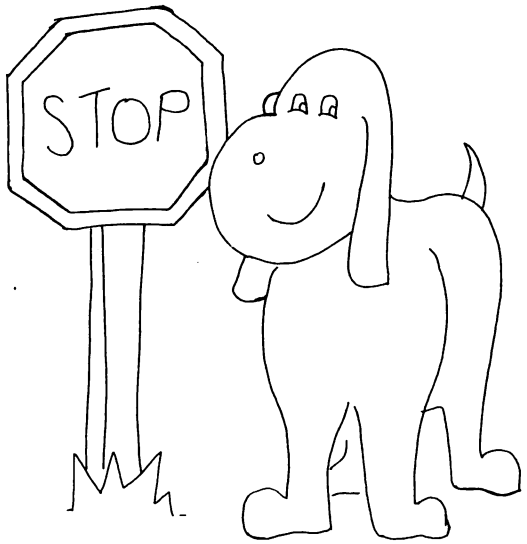
first one is shorter to say and write.

SIZE is known as a *variable*. It represents the length of the length of the side of the square that Sam will draw for this procedure. The word *variable* means that we will be able to change the value of SIZE when we tell Sam to SQUARE. A variable is something whose value can be changed.

The statement :SIZE (we say "dots SIZE" remember!) will allow Sam to adjust the size of the square that he will draw for us. Sam will take the number that you give him for SIZE and use it in his calculations. In the REPEAT command for instance we told Sam to FD :SIZE. When Sam sees this he will go forward the number of steps that SIZE represents.



Square 50.



:SIZE really means the "value of the variable size"

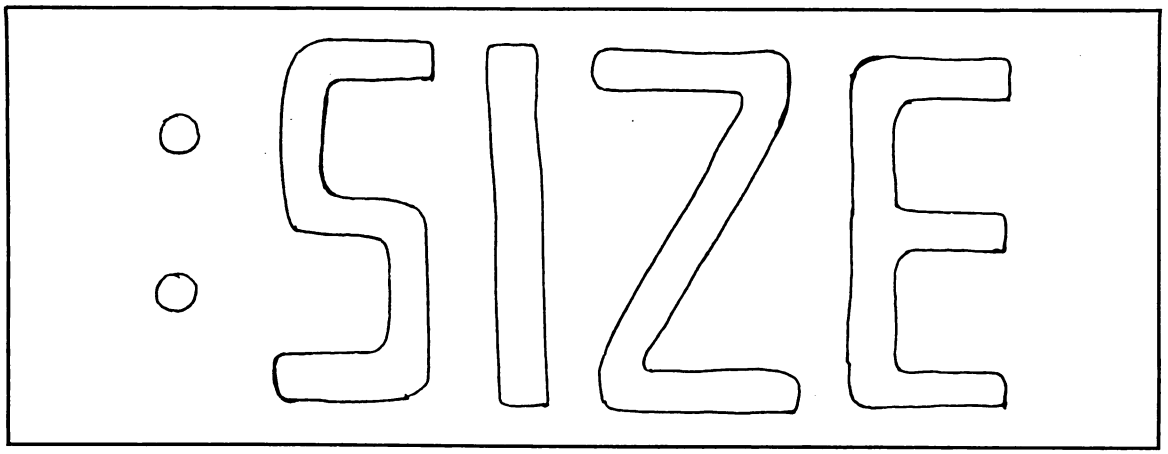
Now that we have seen a procedure that uses a variable, let's look more closely at what a variable is.

WHAT IS A VARIABLE?

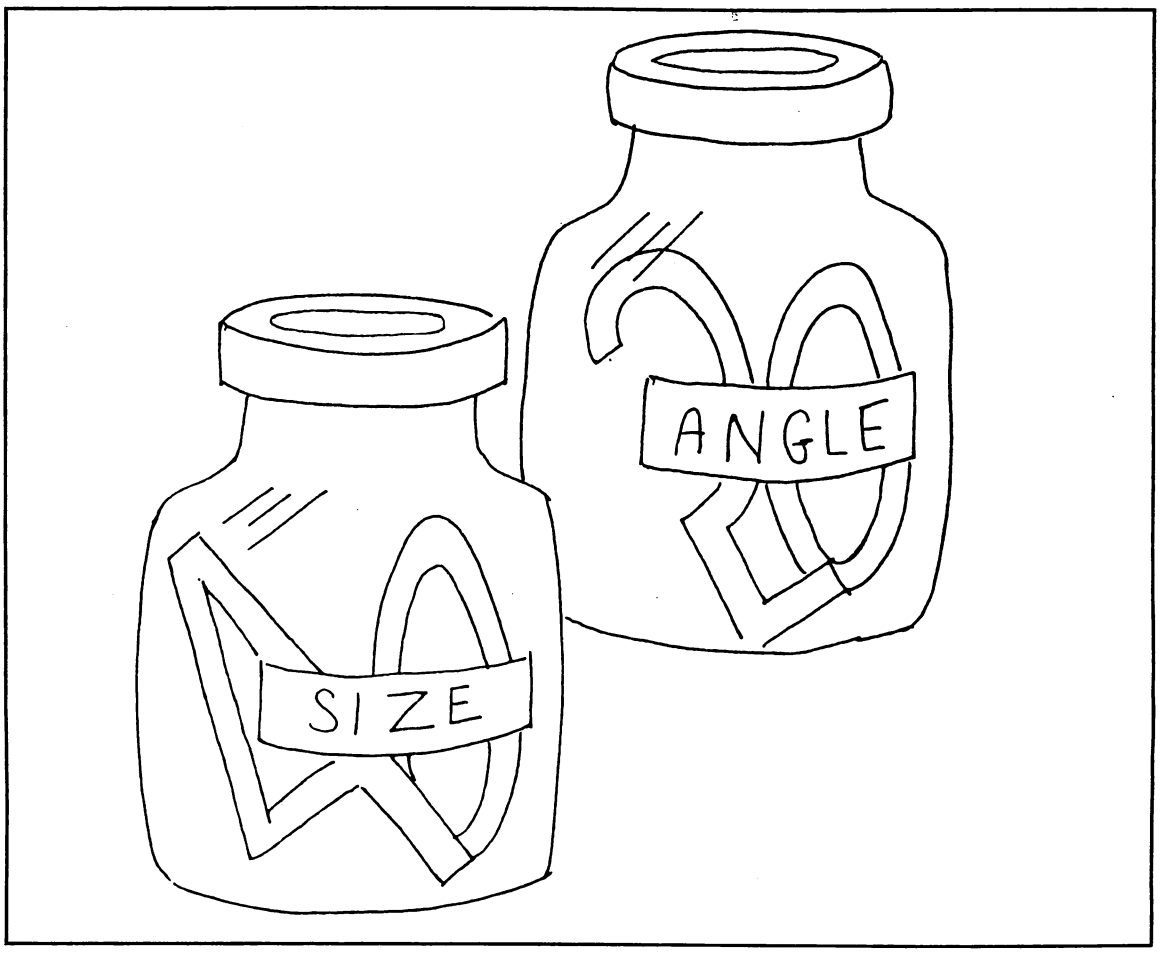
A variable is a Logo word that we use in place of a Logo object. That object can often be a number. For example, you may not know exactly how much you want Sam to go forward, so you simply tell him to FD :SOME, so that you can give Sam the value of SOME later on. You can change the value of SOME as often as you want to.

You can think of a variable in Logo in the following way. Every variable is like a container. Each container has a label name pasted on its side. The container *could* be the type of jar that you keep cookies in!

Everybody knows that bears and turtles love cookies! When Sam finds a container, he will look inside to see what is in it. The thing in the container is the value of the variable.



:SIZE.



Cookie jars hold our numbers.

The contents of the container, like the contents of a cookie jar, can change from time to time. For instance, there might be 5 cookies in the variable—sorry, jar—when a procedure begins, but when the procedure ends, there might be 10, or 12 or even none!

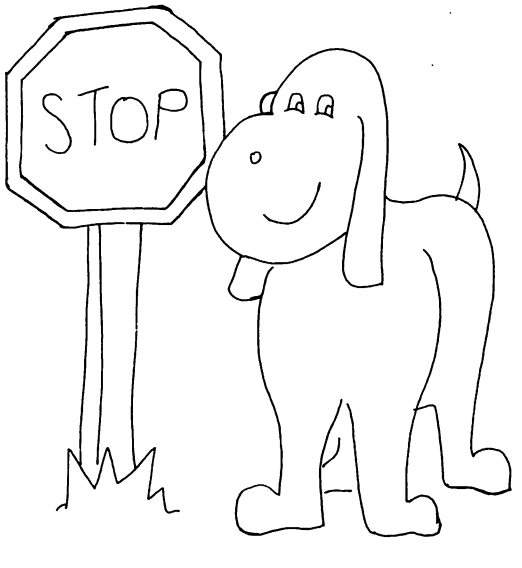
A variable in Logo often *belongs* to its own procedure. This means that different procedures can use the same variable names to mean different things.

Suppose we call a variable COOKIE. If we then say

```
BOX :COOKIE
```

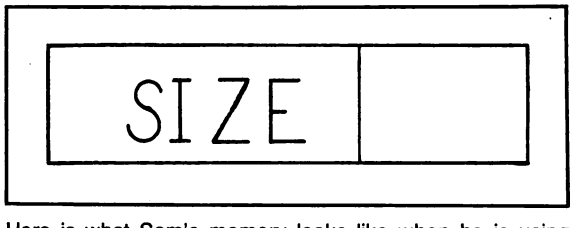
Sam will go to the variable called COOKIE and on seeing the colon (:) (we say "dots" remember?), he will say to himself. "I need the thing in the COOKIE jar". He will then use the number that he finds in the cookie jar to decide on the length of the box in the BOX :COOKIE procedure.

So what is the difference between the Logo words COOKIE and :COOKIE? The word COOKIE is the *label* on the jar. The statement :COOKIE is the thing *inside* the jar.



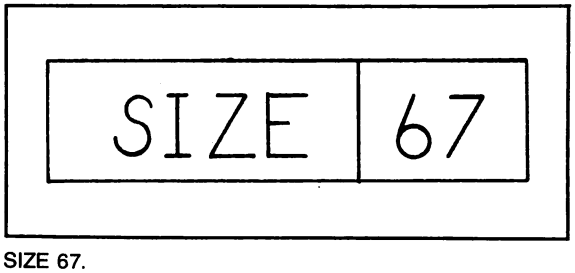
Here is another useful way of thinking about variables, either in Logo or in computing in general. It has been said that if you understand how a computer uses variables, then you understand the computer!

In a computer, a variable is actually held in a little memory cell. Each memory cell has an address number so that the computer can keep track of each one of the cells. Instead of using numbers to label the memory cells, you can label them with a name. The name that you use is the name of the variable! To make this clear, you can draw little boxes to keep track of what is happening. It is common to draw many boxes for each procedure that you write, so that you can follow what is happening to the numbers in the procedure. Here is what the variable :SIZE would look like if we drew it on paper as a "memory box".



Here is what Sam's memory looks like when he is using variables.

If you tell Sam that the value of SIZE is 67, by typing **SQUARE 67** say, this is what your diagram will look like.



SIZE 67.

If you now tell Sam to run the procedure again with a size of 32 by typing **SQUARE 32**,

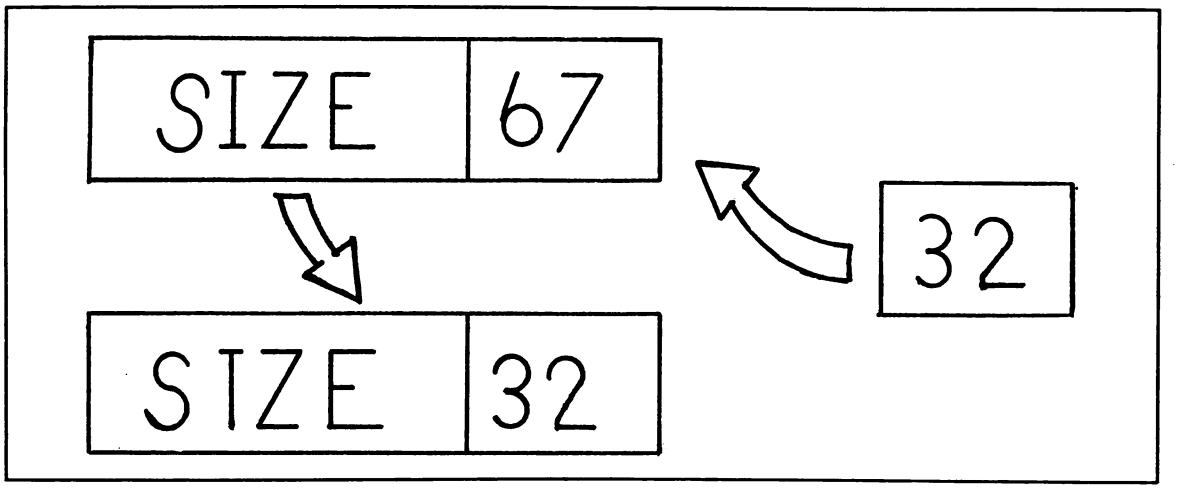
Sam likes variable and cookies.

he will put 32 into the location called SIZE. The old value of 67 will be rubbed out and lost from Sam's memory. Every time you run the procedure with a different value, the old value of SIZE will be replaced by the new value.

Let's see how we can use some more variables in Logo.

A TRIANGLE OF ANY SIZE

Now that you know how to teach Sam about drawing a square of any size, it might be a good idea to teach him to draw a TRIANGLE of any size. You should try to get him to do this without looking at the solution that follows, but if you are stuck, you can look at the solution



SIZE 67 becoming SIZE 32.

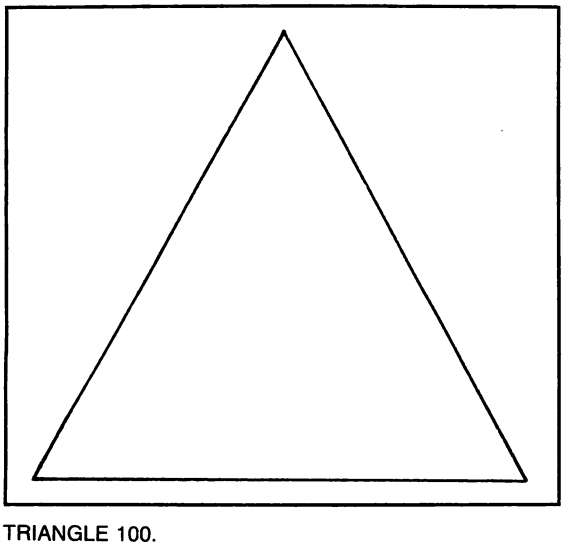
given. Remember, if you did not get exactly what you wanted the first time, simply try again. Sam will carry out your instructions to the letter and each time he does something unexpected, you will be able to learn a little more about Logo and about how to program a computer.

Here is a possible solution that is based on the ideas used in the SQUARE procedure in this chapter and in the TRIANGLE procedure in a previous chapter.

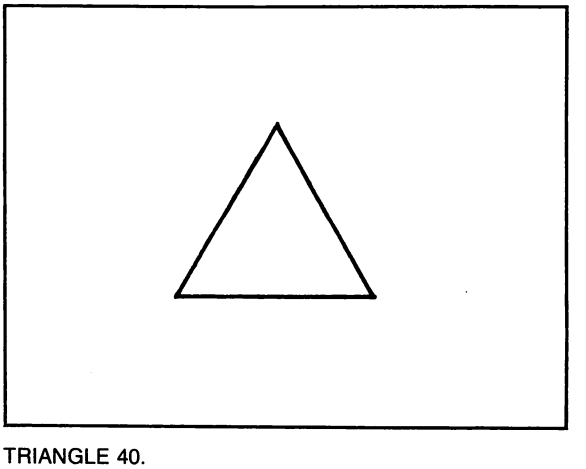
```
TO TRIANGLE :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

To see Sam draw the triangle, type TRIANGLE 100. When he has finished, try typing TRIANGLE 40 and so on.

We didn't use the word COOKIE for the variable this time. We used the word SIZE instead. The word SIZE was probably a better choice for this procedure. If you can choose the name of the variable so that it somehow describes what is happening in the procedure,



TRIANGLE 100.



TRIANGLE 40.

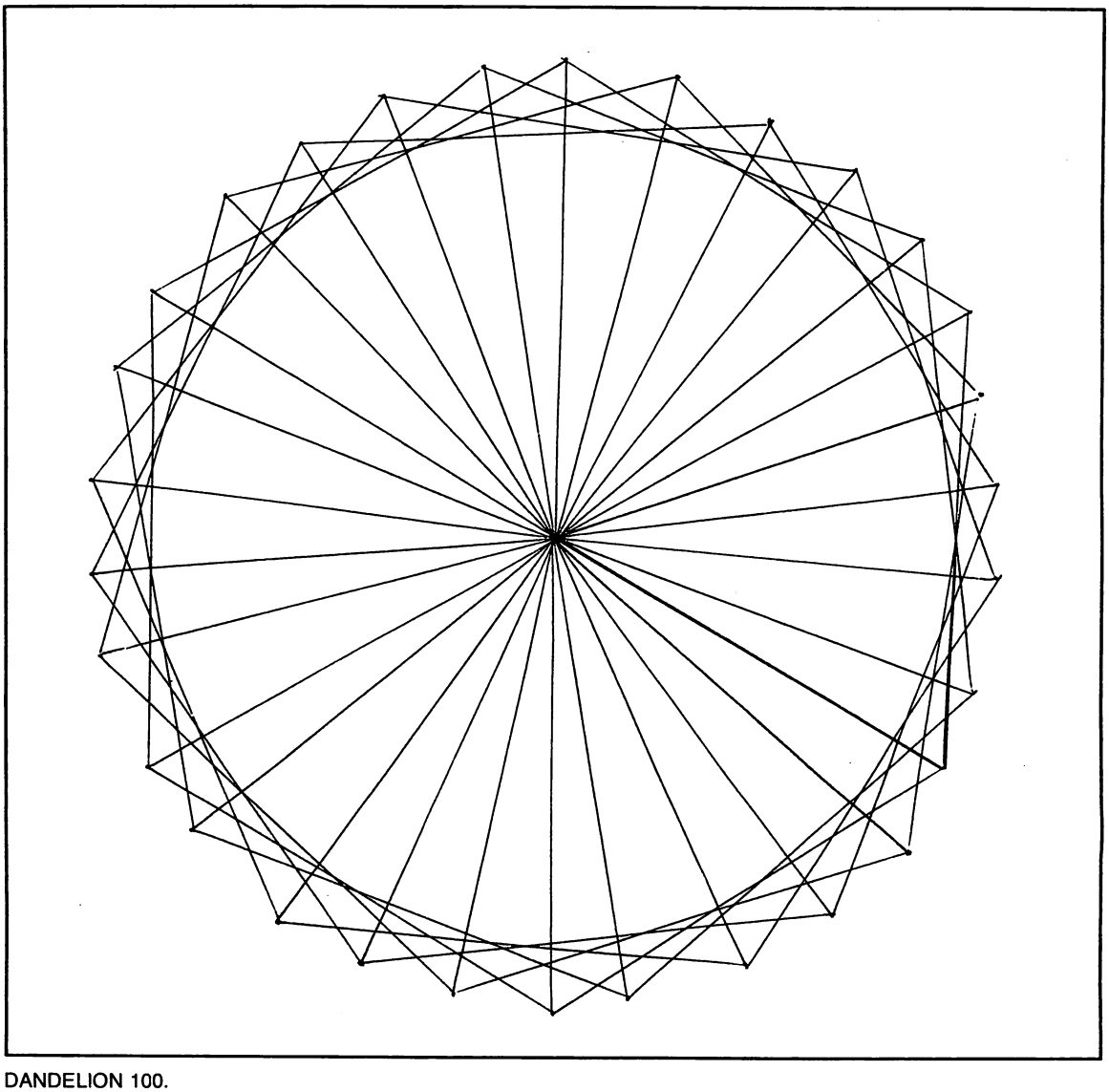
then that is best! For instance, if you want to tell Sam about the value of a radius, then you could use the variable name RADIUS to describe the changing values. Or, if you wanted to tell Sam about how often something has happened, then you could use the variable name TIMES or NUMBER.

If you want to talk about a second radius, then you could use the variable RADIUS2, so that Sam doesn't confuse the first radius with the second radius.

I wonder if we can draw something more complicated using variables?

A DANDELION OF ANY SIZE

Do you remember the dandelion that we drew earlier? I wonder if we could draw a dandelion like that one, but this time, let's

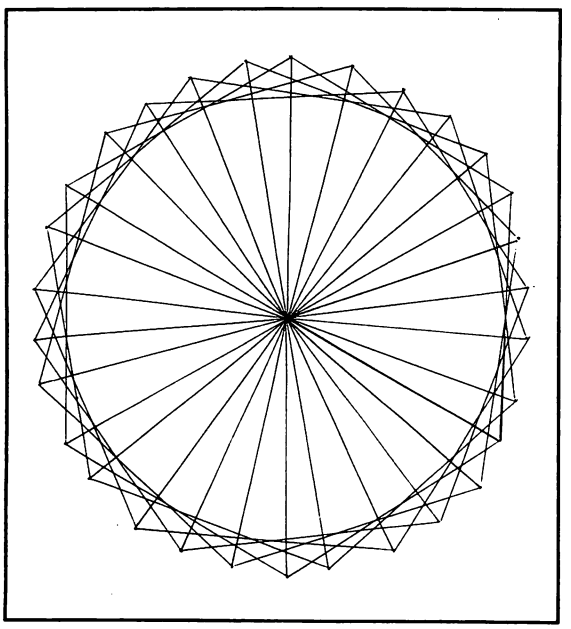


DANDELION 100.

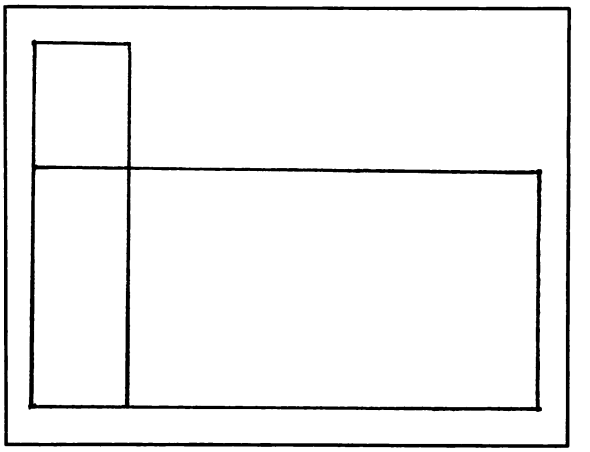
arrange it so that we will be able to tell Sam about the size of the dandelion that we want. How would we do this? Try a solution by yourself, and then look at the solution below. Again, your solution does not have to be the same as the one given. The really important thing is that you try some of your own ideas.

```
TO DANDELION :SIZE
REPEAT 36 [TRIANGLE :SIZE RT
10]
END

TO TRIANGLE :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

When you have typed in the procedure, try typing **DANDELION 100** to see what Sam will do. How about following that with **DANDELION 10** to draw a small diagram! Did you get what you expected for DANDELION 10 or is 10 too small for this procedure?



DANDELION 10.



RECTANGLE 100 50 and RECTANGLE 60 120.

This is an important example! Did you notice that the TRIANGLE procedure has the variable SIZE associated with it? The value for the procedure TRIANGLE has to be passed to that procedure from the main procedure that we have named DANDELION. This is done by using REPEAT 36 (TRIANGLE :SIZE RT 10]. The value of SIZE is transferred by using **TRIANGLE :SIZE**. Try leaving out the :SIZE after TRIANGLE to see what happens! The procedure will not work at all without the variable being transferred properly! This is because the variables used are *local* to the procedures. That is, each procedure has its own set or *library* of variables that it uses. Variable values must be transferred from library to library when the variables are local.

USING TWO VARIABLES

We have looked at some procedures that will allow you to supply a number to a procedure. The procedure will then take the number and use it in it's primitives. Can we arrange it so that we can give the procedure *two* variables to use?

Here is how to have a procedure use two variables. Type the following into the computer.

```
TO RECTANGLE :WIDTH
:LENGTH
REPEAT 2 [FD :WIDTH RT 90 FD
:LENGTH RT 90]
END
```

To see what this procedure does, you will have to supply it with two values, one for the length and one for the width. To make Sam draw a rectangle 100 steps wide and 50 steps long, type the following.

```
RECTANGLE 100 50
```

Then try RECTANGLE 60 120.

CHANGING VARIABLE VALUES WITHIN A PROCEDURE

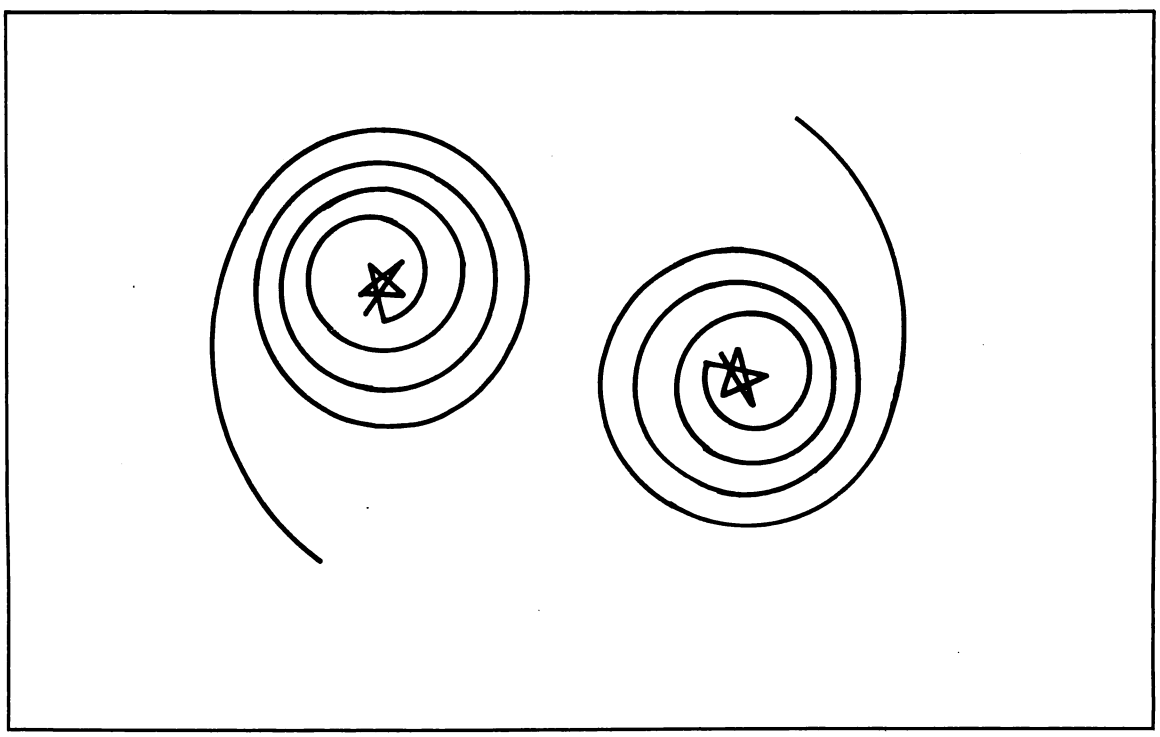
Variables are values that can change within a procedure. Let's see if we can change the values that we give to Sam *within* the same procedure.

Type in the following procedure; it will change the value of the variable DISTANCE within the procedure.

```
TO SPIRAL :DISTANCE :ANGLE
REPEAT 500 [FD :DISTANCE RT
:ANGLE MAKE "ANGLE
  :ANGLE + 2]
END
```

The procedure uses a REPEAT primitive with a value of 500. The large value, 500, was chosen to allow Sam lots of repetitions. The number could easily have been 200 or 600. Any large number will do.

The statement MAKE "DISTANCE :DISTANCE + 2 is the statement that will change the value of the variable DISTANCE. The unusual thing, is the quotation marks in



SPIRAL 20 90.

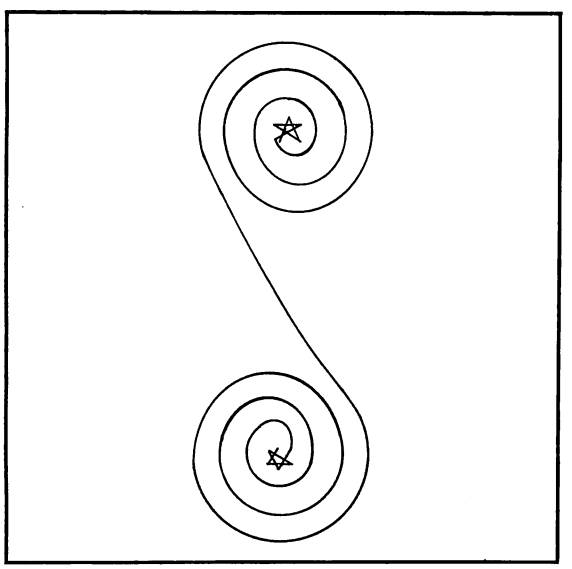
front of the word DISTANCE. In Apple Logo, the quotation mark is used to tell Sam that the word that follows is a *label* for a variable. This means that the word DISTANCE will be pasted on our "cookie jar." On the other hand, :DISTANCE is the *number* in the "cookie jar".

To see Sam execute the procedure, type SPIRAL 10 90, and then try SPIRAL 10 50, to see what happens. The effects are quite different! The procedure produces a spiral. Many interesting patterns can be generated from this and similar procedures.

The value of DISTANCE that we gave to Sam in SPIRAL will increase by a value of two each time the MAKE primitive is used. You should try changing the procedure so that the values will change by 4 each time instead of by 2. What will this give us for a pattern?

Now that we know how to change variable values within a procedure and how to transfer variable values to other procedures, let's look at some more ambitious projects.

In the procedures that follow, we will usually start with English listings of what we want to do. We will follow these listings with the details of the subprocedures, and finally, we will list the complete procedure. This final procedure is what you would have to type to make the procedure work after typing ERPS at the keyboard or after starting the Logo language from the beginning.



SPIRAL 5 50.

A BETTER FLOWER FOR SAM TO DRAW

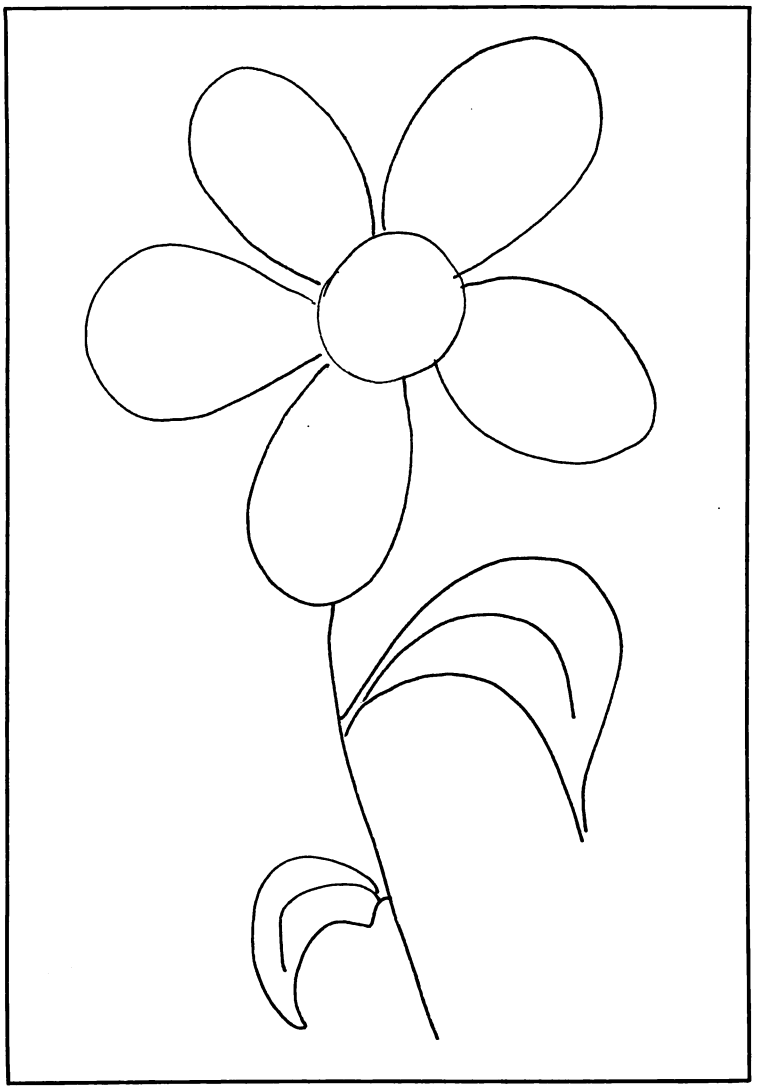
Sam's favorite color is green. He likes grass of course as all turtles do! Sam also likes flowers! I wonder if we can draw a better flower with Sam? Let's see how we might do that.

I am sure that he will be delighted if we teach him how to draw a better flower. We will use this procedure later on in the book to make a daisychain, something else that Sam likes a lot! Let's look first of all at a real flower, and then let's look at an English list of what we would have to do to draw that flower. Page 132 shows an illustration of a real flower.
It is very pretty. Sam thinks so too. Here is how we could draw a flower like it.

```
Flower.
Draw the stalk of the flower.
Draw the head of the flower.
End.

The head of the flower.
Draw all of the petals in the head.
End.

Petals.
Draw a petal.
End.
```

We have used the idea of subprocedures again as you can see. This makes it so much easier for us and of course, if it is easier for us, then it is easier for Sam.

The flower is made up of two parts, the stalk and the head. The head in turn is made up of petals. Let's use variables in the procedure



A "real" flower.

so that we can decide on the size of the flower when we give Sam the command to FLOWER. This means that we will be telling Sam to FLOWER 100, or FLOWER 200 when we eventually tell him to draw the flower.

Here is the controlling procedure.

```
TO FLOWER :SIZE
STALK :SIZE
HEAD :SIZE
END
```

All of the procedures have a variable SIZE attached to them. This means that we will be able to transfer the variable SIZE from one procedure to another. Of course, we can't use the same dimension for the head and the stalk and the petals. The petals for instance will not be the same length as the stalk. We will have to adjust the length value within each procedure, so that each part of the flower will be the correct size in relation to all of the other parts of the flower.

Here is the procedure for drawing the stalk.

```
TO STALK :SIZE
ARCR :SIZE 50
END
```

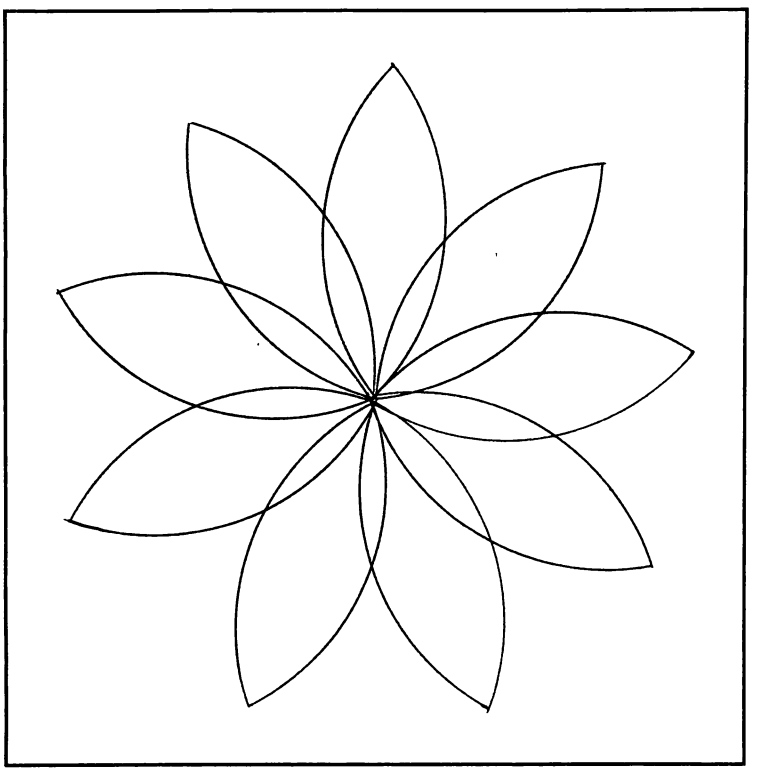
There is our old friend ARCR. Of course Sam knows how to draw an arc all by himself! Can you remember what the value 50 stands for. It stands for 50 degrees of arc. If you remember, there are 360 degrees of arc in a complete circle, so that 50 degrees of arc is only a small part of a complete circle. In fact, it is slightly less than one sixth of a circle, as 60 degrees of arc is exactly one sixth of a circle!

Here is the procedure for drawing a petal.

```
TO PETAL :SIZE
ARCR :SIZE*0.25 80
RT 100
ARCR :SIZE*0.25 80
END
```

To see what the petal will look like, try typing PETAL 100.

There are more ARCs in the petal procedure, but what is this about :SIZE*0.25? First of all, the symbol * means multiply in Logo. Every time that Sam sees a * between two numbers, he will try to multiply them. This is the part of the procedure in which we will reduce the value of SIZE to produce a petal of a suitable size. If we give Sam the value 100 for SIZE, the stalk would probably be the right size, but for our little petals it would be too big. If we multiply :SIZE by 0.25, we will get a new value of 25 for our petal length. That should be just about right! In case you have forgotten, 0.25 is the same as one quarter, so 100*0.25 is



HEAD.

the same as 100 times 1/4, which of course is the same as 100 divided by 4, which is 25.

Now that Sam knows about our petal, let's look at how we will teach him to draw the head of the flower using the fact that he knows how to draw petals.

Here is the procedure to draw the head of the flower.

```
TO HEAD :SIZE
REPEAT 9 [PETAL :SIZE RT 60]
END
```

Now let me see. . .nine times 60 comes out to 360. There is that number again!

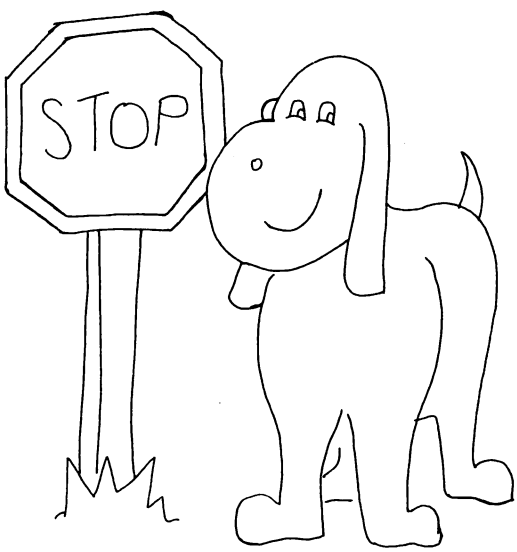
To see the head of the flower, try typing HEAD 100. You can try each subprocedure as you go along. By doing this, you will be able to tell if your subprocedures are working properly. If your subprocedures are working well, your main procedure should also work well.

That covers all the parts of the flower. All that we have to do now is to put all of the parts together and type them into the machine. Here is what you need to type to see Sam draw a flower. You get to choose the size of the flower. Remember to clear out Sam's memory by using ERPS before you start typing, if it is needed!

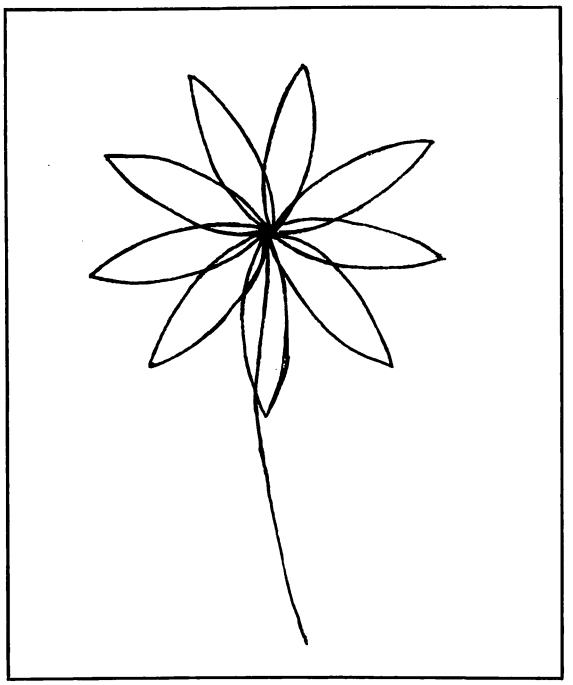
```
TO FLOWER :SIZE
STALK :SIZE
HEAD :SIZE
END

TO STALK :SIZE
ARCR :SIZE 50
END

TO HEAD :SIZE
REPEAT 9 [PETAL :SIZE RT 60]
END

TO PETAL :SIZE
ARCR :SIZE*0.25 80
RT 100
ARCR :SIZE*0.25 80
END
```

To see Sam draw the flower, try typing FLOWER 100. After that, try typing FLOWER 200. How small is the smallest flower you can draw with the procedure? Are little flowers best, or are big flowers best?

Can you change the procedure to change the color for the background? How about changing the procedure to draw a lollipop instead of a flower, so that when you type LOLLIPOP 100 you get a large lollipop, but when you type LOLLIPOP 20, you get a small lollipop!



In many of the following procedures, you will find two special subprocedures. They are SET and MOVE. The SET procedure will tell Sam to go to a starting position on the screen at the beginning of a procedure. It will also set



Flower 100.

the pen color, the screen mode, and the state of the pen. The MOVE procedure is sometimes, called MOVE1 or MOVE2, but usually it is just called MOVE. It will move Sam from one drawing location to another in the middle of a procedure. The MOVE procedure will often involve a PU and a PD so that Sam can move to his new position without drawing lines.

Let's teach Sam how to build a *variable* tower!

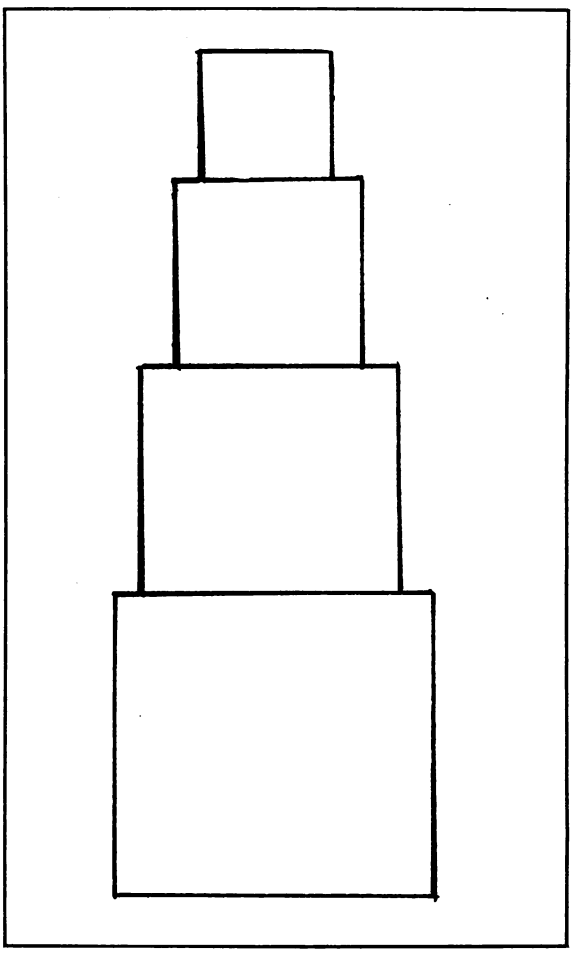
A TOWER OF BLOCKS FOR SAM TO DRAW

Here is a procedure that will draw a tower for us. It uses only one variable and as a result, it is fairly easy to follow. Before we get to the procedure itself though, let's sketch out what we intend to do in English. Here is an illustration of what we have in mind for the tower.

Hmm. . .the blocks that make up the tower are all the same shape. Maybe it will be possible to use one basic square procedure and change a variable so that the squares will become smaller!

Let's keep that in mind as we write the English list of instructions for Sam. Here are the instructions that we need.

Move from the HOME position to the bottom left.
Draw the first square.
Move to the start of the second square.
Draw a smaller version of the first square.
Move to the start of the third square.



Tower.

Draw a still smaller square.
Move to the start of the fourth square.
Draw a still smaller square.
End the procedure.

Hmm. . . there seems to be a lot of those "move" steps. Perhaps we should tell Sam to make the move step a subprocedure. Each time we wanted to move, we would simply call that subprocedure.

Here is the beginning of our procedure. This procedure will control all of the others.

```
TO TOWER
SET
SQUARE 50
MOVE 50
SQUARE 40
MOVE 40
SQUARE 30
MOVE 30
SQUARE 20
END
```

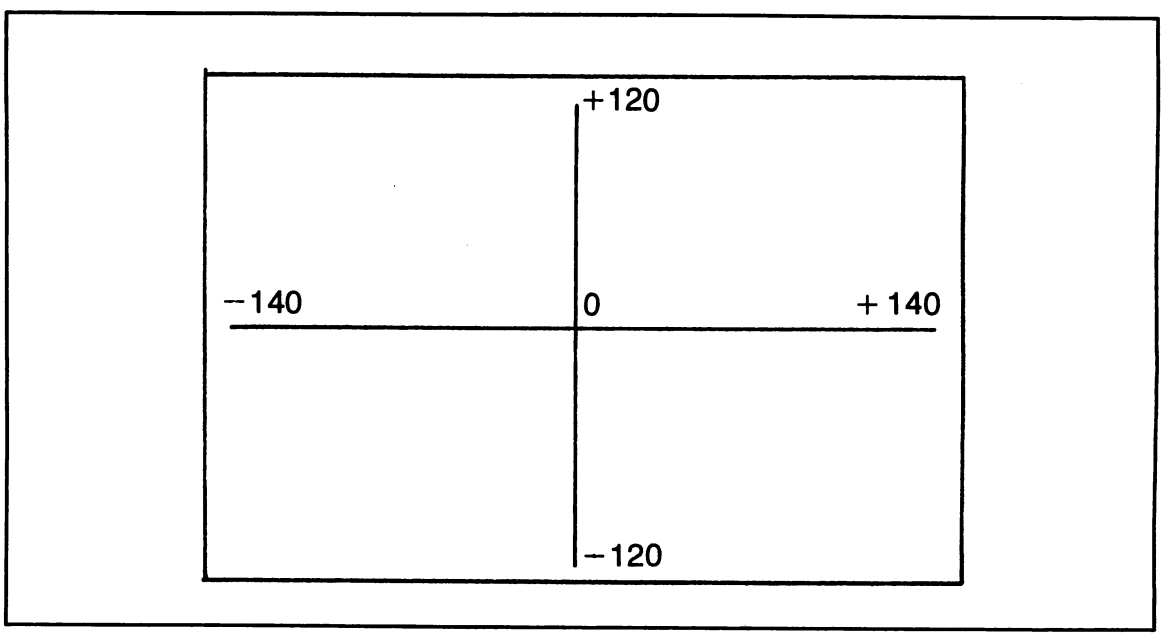
We used the procedure name SET to describe the process of initializing Sam. The procedure will move Sam to his starting point, lower the pen, and set the screen mode.

Here is a procedure that will initialize Sam's position.

```
TO SET
FULLSCREEN
SETBG 4
PU
SETPOS [-30 -110]
PD
END
```

What is the primitive SETPOS? That is a new one for us! SETPOS [−30 −110] will move Sam to position −30 −110 on the screen. From the screen map, you can see that −30 −110 is on the lower left of the screen.

What color would you expect the background to be? Correct, green is one of Sam's favorite colors!



Screen map.

You have seen this next procedure before. It is our old friend the SQUARE. The SQUARE procedure has a variable SIZE to go with it. When the procedure is called from TOWER, the value that follows the procedure call will be transferred to the subprocedure SQUARE :SIZE.

```
TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END
```

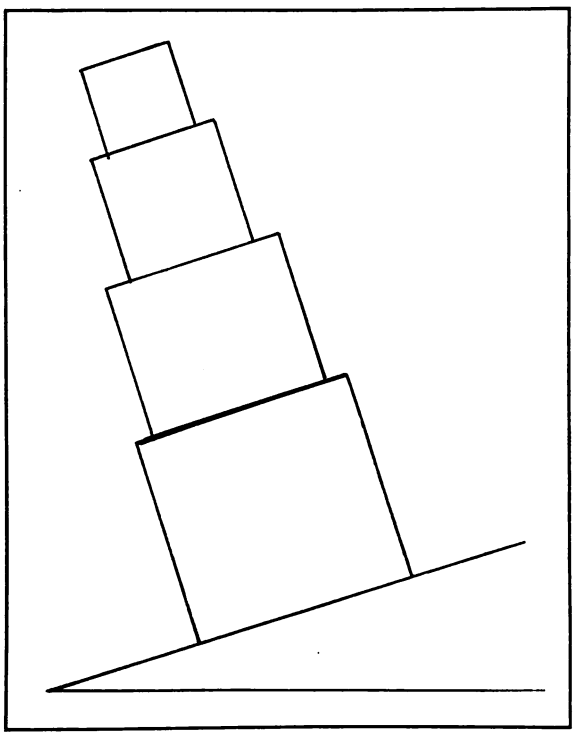
Finally, let's look at the procedure to MOVE to the new starting position for the next square.

```
TO MOVE :SIZE
FD :SIZE
RT 90
FD 5
LT 90
END
```

I bet that you noticed something very important here! The procedure MOVE uses the same variable SIZE that the procedure SQUARE used. The TOWER procedure will not work unless this is so. Sam will have to MOVE forward the same distant as the edge of the last square that he drew, in order to end up at the correct starting point!

Here is the complete procedure all tied together. This is what you must type in to see the TOWER procedure in operation. The order in which you type the procedures does not matter. However, the order of the primitives within the procedure does matter. After all, you wouldn't want poor Sam to get things backwards or in the wrong order would you? It pays to keep Sam organized.

```
TO TOWER
SET
SQUARE 50
MOVE 50
SQUARE 40
MOVE 40
SQUARE 30
MOVE 30
SQUARE 20
END

TO SET
PU
FULLSCREEN
SETBG 4
PU
SETPOS [-30 -110]
PD
END

TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END
```



The leaning tower of Sam.

```
TO MOVE :SIZE
FD :SIZE
RT 90
FD 5
LT 90
END
```

Simply type TOWER to see the procedure. The value of the variable SIZE is set when the subprocedures are called. If the procedure does not give you the diagram shown, check for typing errors, and try again!

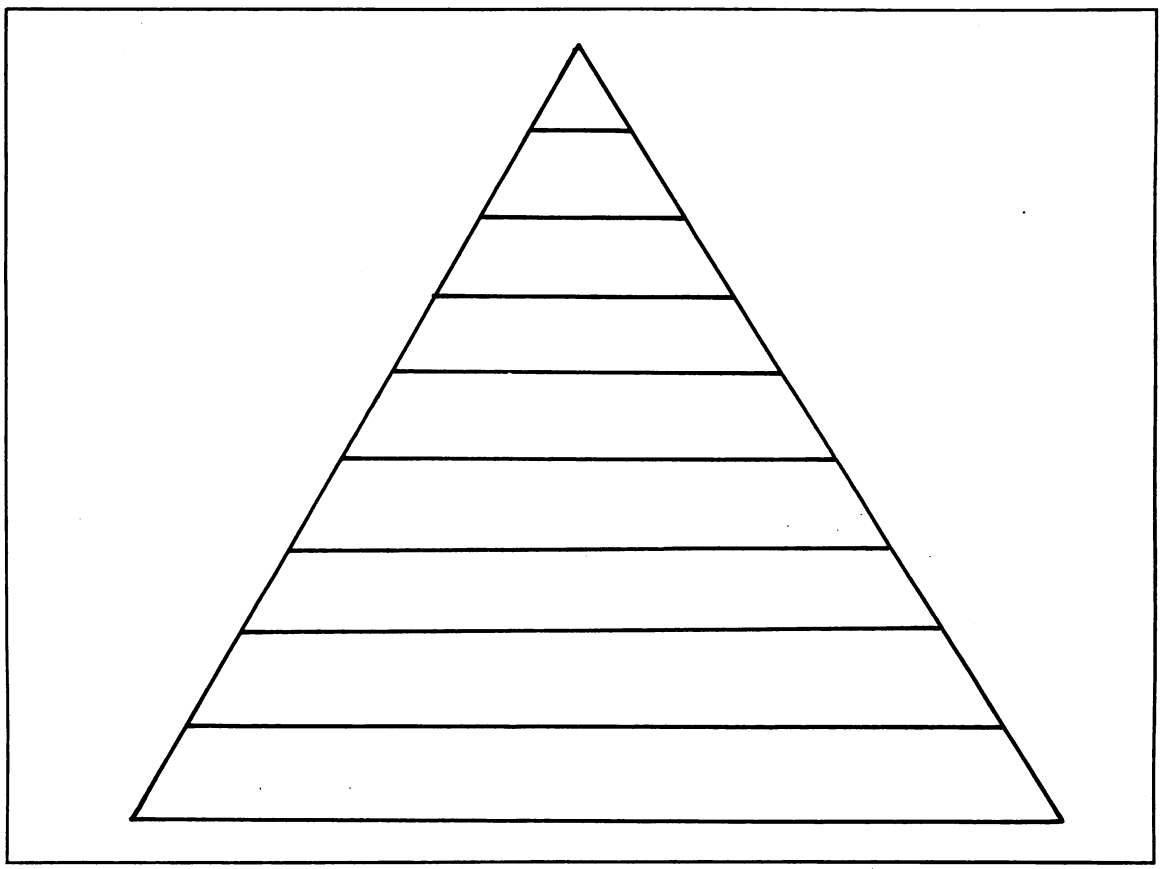
Can you make the tower lean, like the one in the illustration?

Can you rewrite the procedure so that when you type TOWER 50, you get a small tower and when you type TOWER 100, you get a large tower?

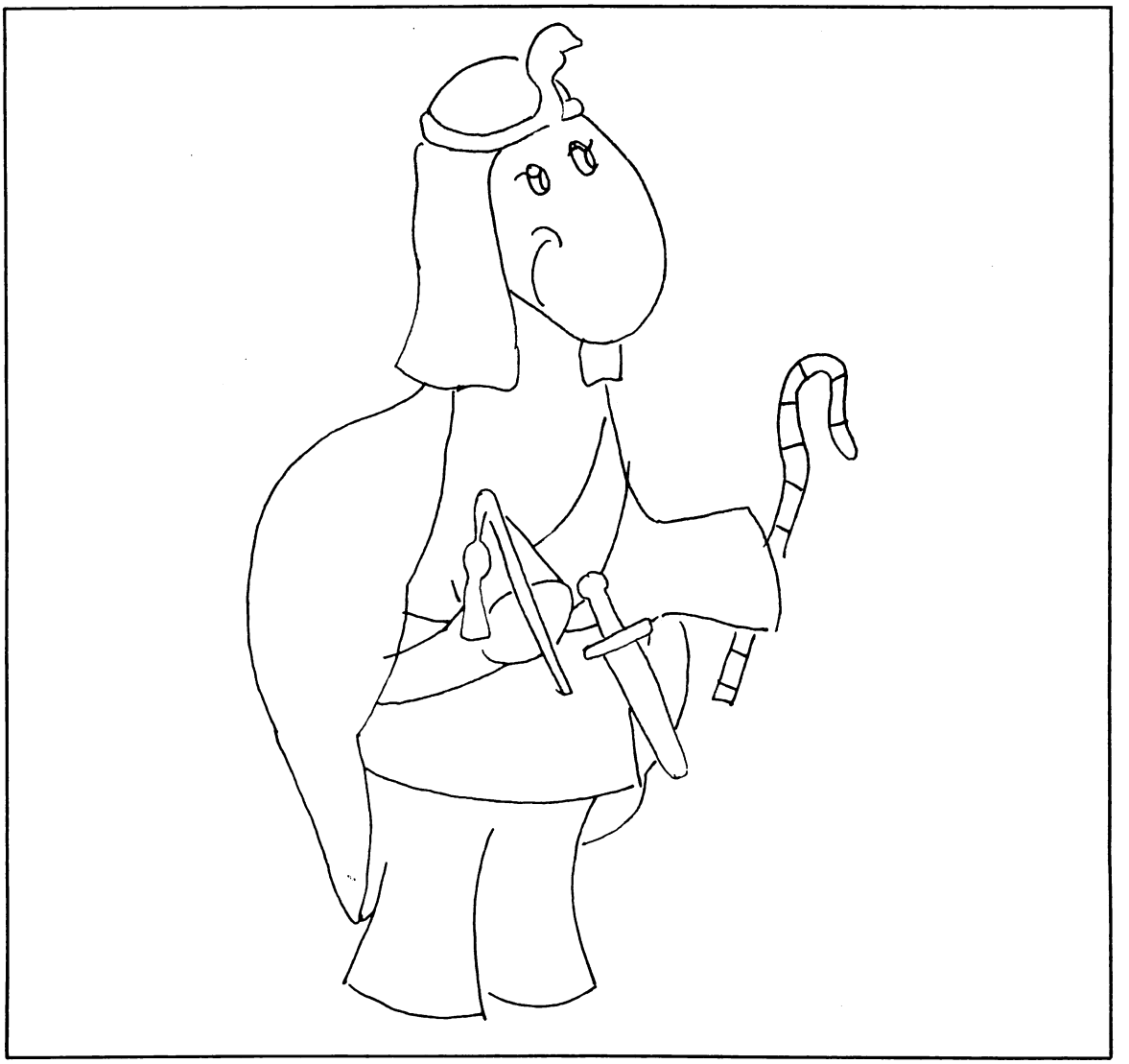
A VISIT TO EGYPT WITH SAM

Now that we have finished with towers, let's move on to pyramids! Sam is a very widely traveled turtle. He is very keen on Egypt, the thing that he likes most about Egypt is the pyramids. I wonder if we could teach Sam to draw a pyramid? Let me see . . . A pyramid is a lot like a triangle. We can base our pyramid procedure on the triangle procedure! Here is a procedure that will teach Sam how to draw a pyramid.

First of all, let's look at the English statements for the pyramid.



Pyramid.



Sam the Egyptian.

```
Pyramid
Move to the top of the pyramid be-
    fore we draw the
    first triangle.
Draw a big triangle.
Draw a smaller triangle.
Draw an even smaller triangle.
. . .
. . .
. . .
Hide Sam.
End.
```

The controlling procedure would probably look like this.

```
TO PYRAMID
SET
```

```
TRI 180
TRI 160
TRI 140
TRI 120
TRI 100
TRI 80
TRI 60
TRI 40
TRI 20
HT
END
```

Guess what the TRI procedure does. Correct, it will draw triangles of different sizes. The size of the triangle will depend on how large or how small the number following TRI is. The procedure SET will move Sam to the top of the triangle. SET will also set the screen mode and the screen color, as well as raise and lower Sam's pen as needed. The HT at the end is simply to give us a nicer picture by hiding Sam at the end of the procedure.

Here is what the SET procedure would have to look like to put Sam at the top of the screen, ready to draw his first triangle.

```
TO SET
FULLSCREEN
SETBG 4
PU
SETPOS [0 100]
RT 150
PD
END
```

Did you notice that we have PU as the fourth primitive and PD as the last primitive? We needed the PU to keep Sam from drawing all over our screen as he moved into position. If we left out the PD at the end of the procedure, Sam wouldn't be able to draw when it came time for him to TRI! Here is the procedure to draw triangles.

```
TO TRI :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

The procedure TRI is one that we have seen before. This one has a variable SIZE attached to it. It is the variable SIZE that allows us to send Sam numbers for our triangle size. Our second call in the procedure PYRAMID is to TRI 180. The 180 is sent along to the procedure TRI :SIZE, and the value of SIZE becomes 180. This is what allows Sam to draw triangles of different sizes.

Here are all the procedures strung together. This is what you would type in at the beginning of a session to see Sam draw a pyramid. If you have already started your computer, it might be a good idea to type ERPS to clear Sam's workspace.

```
TO PYRAMID
SET
TRI 180
TRI 160
TRI 140
TRI 120
TRI 100
TRI 80
TRI 60
TRI 40
TRI 20
HT
END

TO SET
FULLSCREEN
SETBG 4
PU
SETPOS [0 100]
RT 150
PD
END
```

```
TO TRI :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

Type PYRAMID to see Sam draw an Egyptian wonder!

So far, we have used only one or two variables in a procedure. How many variables can one procedure handle? Let's find out in the next section.

HOW MANY VARIABLES CAN SAM HANDLE ANYWAY?

How many variables can you use with a procedure? You can use as many as you want! Here is a procedure that uses four variables all together. The variables are listed at the beginning of the procedure, and each variable will control the size of a different shape. The different shapes will be drawn one on top of the other to produce a pattern of shapes. If we had wanted to, we could have used only *one* variable and had that variable transfer different sizes to the different shapes; but this way, you can decide on how big *each* shape will be at the beginning of the procedure. Their sizes will not be related one to the other.

Here is the procedure written in English statements.

```
Get all the variable, SIZE1, SIZE2
  etc.
Draw a square of SIZE1.
Draw a circle of SIZE2.
Draw a triangle of SIZE3.
Draw a polygon of SIZE4.
End.
```

Here is the code that you will need.

```
TO MANY.VARIABLES :SIZE1
:SIZE2 :SIZE3 :SIZE4
SQUARE :SIZE1
CIRCLER :SIZE2
TRIANGLE :SIZE3
POLY :SIZE4
END
```

Of course, we would need to supply Sam with procedures for the commands SQUARE, TRIANGLE, and POLY. We don't need to supply him with a procedure for CIRCLER, of course, as he already knows all about circles!

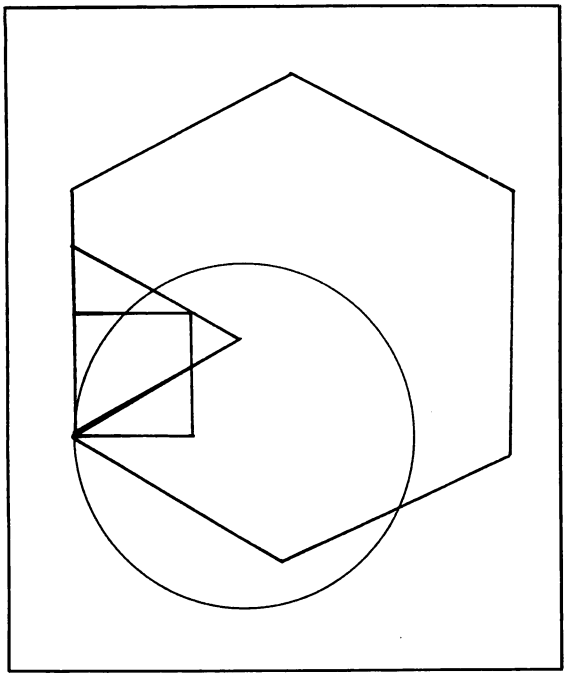
Here are the rest of the procedures.

```
TO SQUARE :SIZE1
REPEAT 4 [FD :SIZE1 RT 90]
END

TO TRIANGLE :SIZE2
REPEAT 3 [FD :SIZE2 RT 120]
END

TO POLY :SIZE4
REPEAT 6 [FD :SIZE4 RT 60]
END
```

Now let me see. . . in POLY :SIZE, we have a six and an RT 60. What is 6 times 60? Why 360 of course. There goes that special number 360 again!

Here is the complete procedure all put together. This is what you would type to see Sam do his tricks with four variables. Remember to type ERPS before you begin if it is necessary.

```
TO MANY.VARIABLES :SIZE1
:SIZE2 :SIZE3 SIZE4
SQUARE :SIZE1
CIRCLER :SIZE2
TRIANGLE :SIZE3
POLY :SIZE4
END
```



MANY.VARIABLE 50 60 100 20.

```
TO SQUARE :SIZE1
REPEAT 4 [FD :SIZE1 RT 90]
END

TO TRIANGLE :SIZE3
REPEAT 3 [FD :SIZE3 RT 120]
END

TO POLY :SIZE4
REPEAT 6 [FD :SIZE4 RT 60]
END
```

Type the command MANY.VARIABLE 50 60 100 20 to see what happens.

After that, try typing MANY.VARIABLES 200 40 80 100, and then try your own combinations of numbers. Can you add color to this procedure? Here is an even bigger challenge; can you separate the triangle, the polygon, the square and the circle, so that they no longer lie on top of each other?

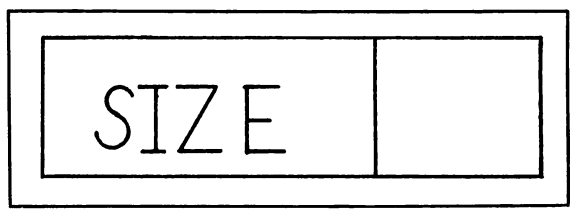
SUMMARY

Here is a summary of the things covered in this chapter so far.

1. You can tell Sam to use variables instead of numbers. This allows you to send Sam information about how you want him to do things.
2. You can use variables in the Logo procedure SQUARE by opening the procedure with the Logo command

   TO SQUARE :SIZE

3. Variables are like containers in Logo. The container has a name, for example, "SIZE, and the variable in that container is :SIZE.
4. In Logo we say :SIZE as "dots SIZE.'
5. It is sometimes useful to think of variables being stored in computer memory boxes like this.



Memory box.

6. If a new value is placed in a variable, then the old value is lost.
7. In Logo, you can change the value of a variable within a procedure by using the MAKE primitive. MAKE "SIZE 34 would place the value 34 in the container SIZE.
8. In Logo, values for local variables must be transferred from one procedure to another.
9. In Logo, an * means multiply.
10. Values of a variable can be changed within a procedure by using primitives like CIRCLER :SIZE*2.

11. Sam can handle many variables in a single procedure.

CHALLENGES

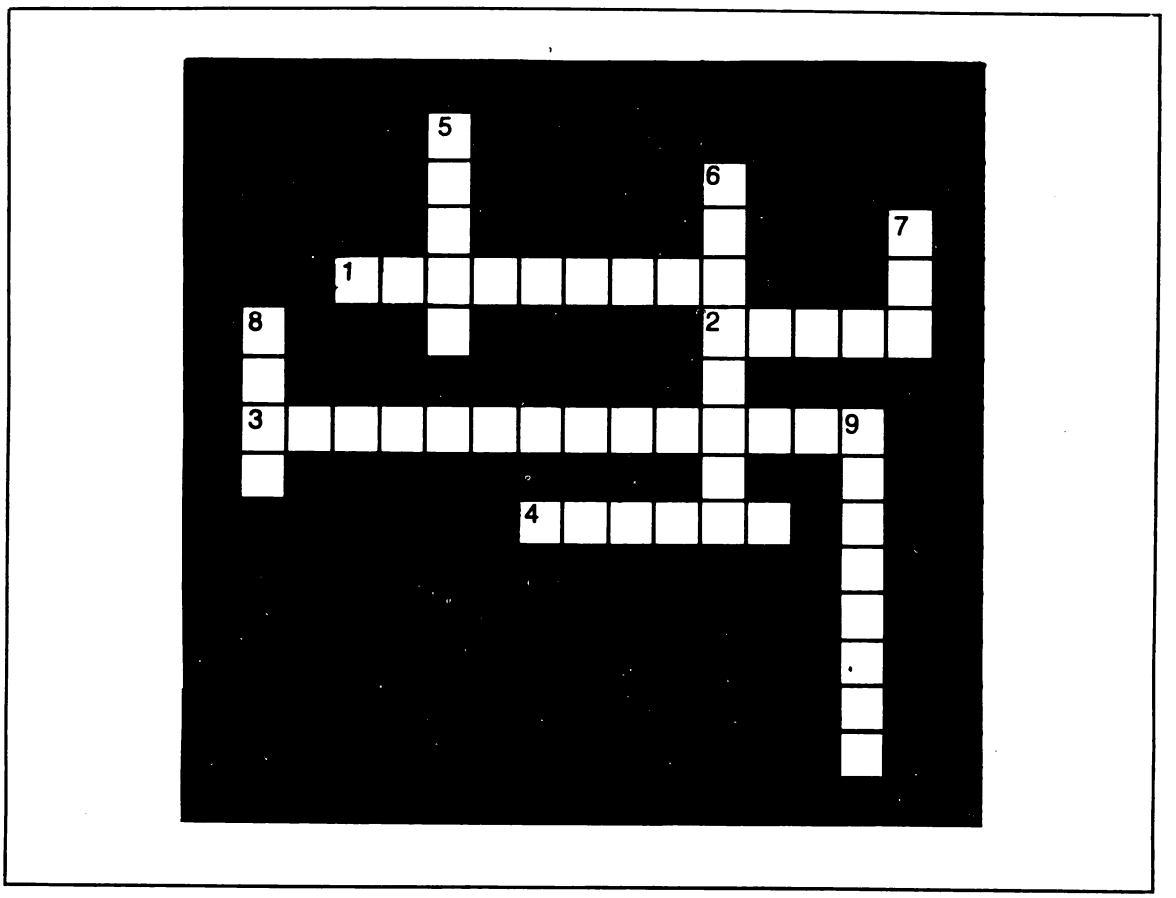
1. Use variables to draw a circle with a radius of the user's choice.
2. Use variables to draw a triangle pattern in a size of your choice.
3. Use a variable to draw a top hat in a size of the user's choice.
4. Use a variable to draw a rectangle of your choice.
5. Use a variable to draw a group of squares of ever increasing size.
6. Use variables to draw an umbrella in a size of your choice.
7. Use variables to draw your initials in a size of your choice.

QUIZ

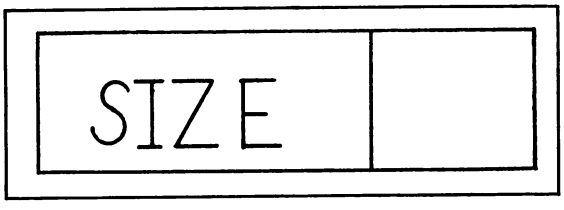
1. You can tell Sam to use _____ instead of simply numbers. This allows you to send Sam information about how you want him to do things.
2. You can introduce variables into the procedure SQUARE by opening the procedure with the Logo command

   TO SQUARE _____

3. Variables are like _____ in Logo. The container has a name; for example "SIZE, and the _____ in that container is :SIZE.



4. In Logo we say :SIZE, as "_____ SIZE.'
5. It is sometimes useful to think of variables being stored in computer _____ like this.



Memory box.

6. If a new value is placed in a variable, then the old value is _____.
7. In Logo, you can change the value of a variable within a procedure by using the _____ primitive. MAKE "SIZE 34 would place the value _____ in the container _____.
8. In Logo, values for variables must be _____ from one procedure to another.
9. In Logo, an * means _____.
10. Values of a variable can be _____ within a procedure by using primitives like CIRCLER :SIZE*2.
11. Sam can handle _____ variables in a single procedure.

CROSSWORD PUZZLE

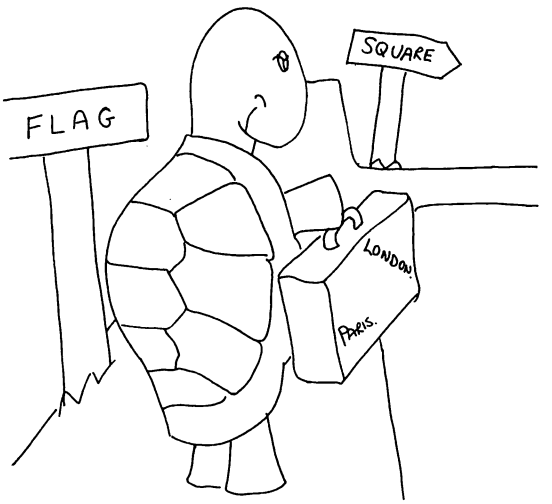
Clues Across

1. A variable is a lot like one of these.
2. If we give Sam a number to use, we call it an _____.
3. Variables used in Logo are said to be _____.
4. We put this in a variable.

Clues Down

5. Another name for the symbol : is _____.
6. Sam likes to change this!
7. When we see the symbol :, we say _____.
8. We can _____ a subprocedure from within a main procedure.
9. This command will tell Sam to draw a square with each side 65 turtle units long.

Chapter 8



Telling Sam, to Tell Sam, to Tell Sam . . .

Have you ever seen a dog chase his tail? He will keep chasing for a long time, and he seems to enjoy it! Did you know that Sam can chase his tail? When Sam is chasing his tail, we use the word *recursion* to describe what is happening inside the computer.

Recursion is an important idea in computing! What is recursion? Recursion occurs when a procedure calls itself. Up to this point, procedures have always called other procedures. What happens if a procedure calls its own name? If a procedure calls its own name, it is said to be making a *recursive call.* This recursive call often occurs just before the procedure's END statement. Here is what a recursive procedure looks like.

```
TO CIRCLE    RECURSIVE CALL
FD 1
RT 1
CIRCLE
END
```

An outline of a recursive procedure.

You know of course that we talk about *telling* Sam when we give Sam something to do. When we use recursion, we are telling Sam to tell Sam to tell Sam to tell Sam and so on and so on.

Recursion will often produce endless loops, which will keep going forever. We sometimes call these *infinite* loops.

RECURSIVE SQUARES

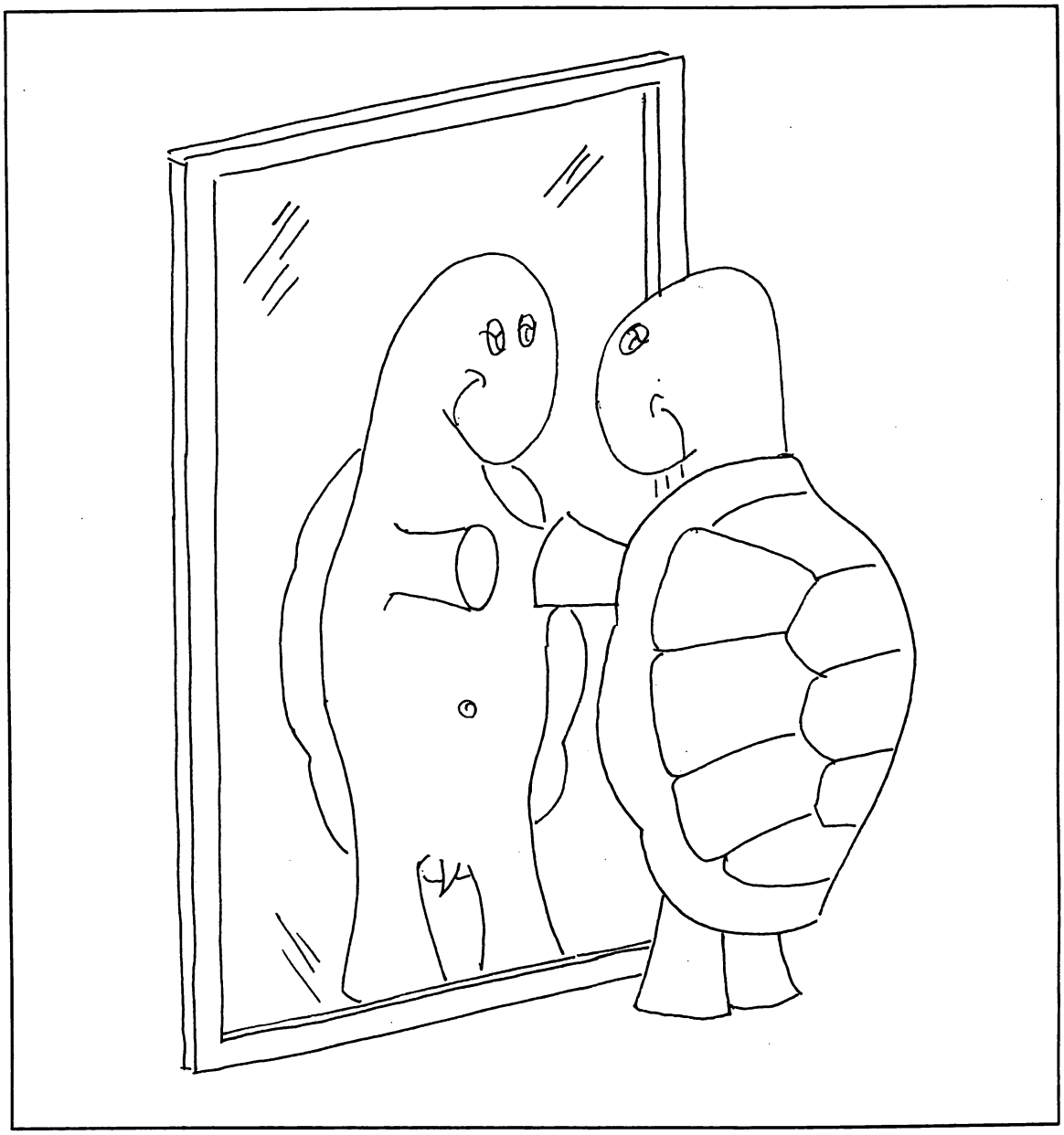
Here is a procedure called SQUARE, which has the procedure SQUARE inside itself.

```
TO SQUARE
FD 50
RT 90
FD 50
RT 90
```

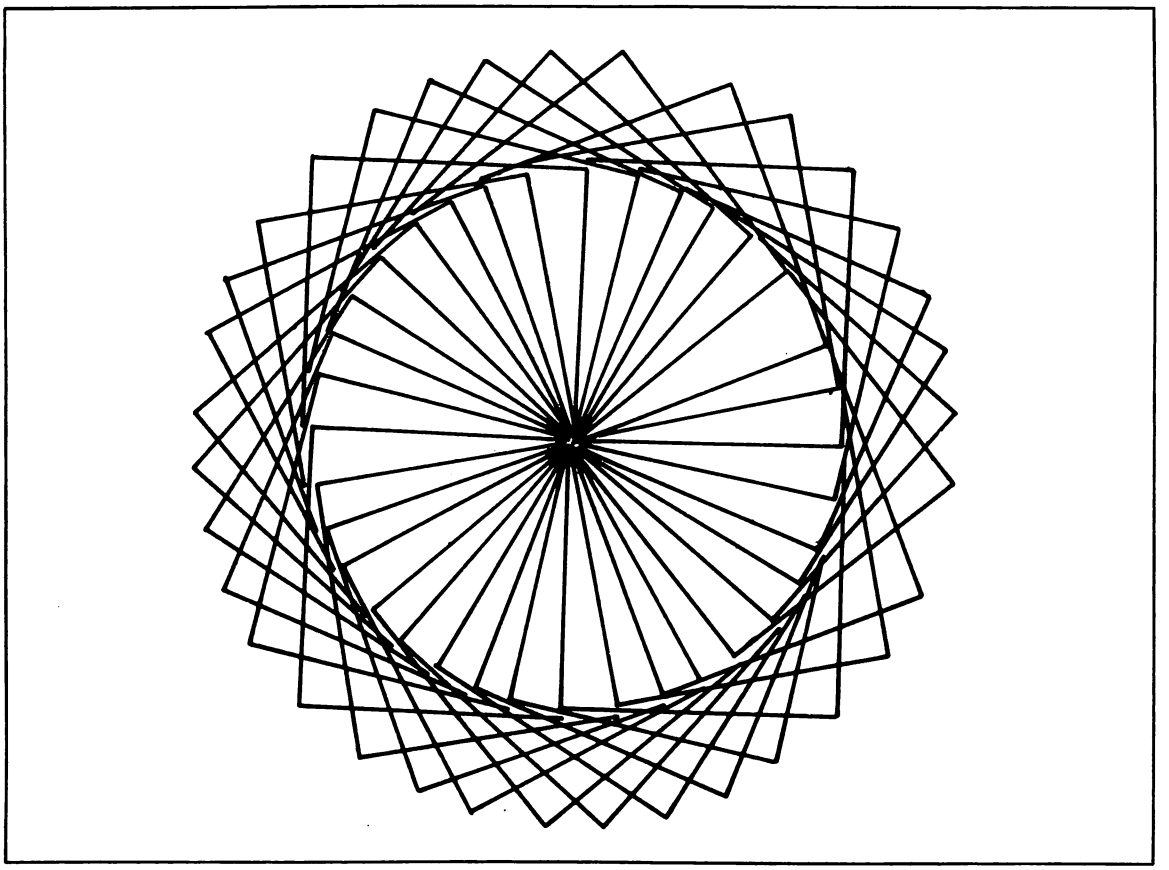
```
FD 50
RT 90
FD 50
RT 80
SQUARE
END
```

Type in the program and then type SQUARE to see what it does. You should see the design on page 147.

What happened was that Sam did the first SQUARE, and then went back to the beginning and carried out the procedure SQUARE again.



Sam telling Sam.



SQUARE.

When he came down through the procedure for the second time, he got to the command SQUARE again. He then went back to the beginning of SQUARE once more. Of course this means that he will keep on doing this forever, because after all, we didn't tell him when he should stop! In fact, if you have typed in the procedure and are sitting beside your computer, you will perhaps be wondering how to stop poor Sam! You can stop the program by typing

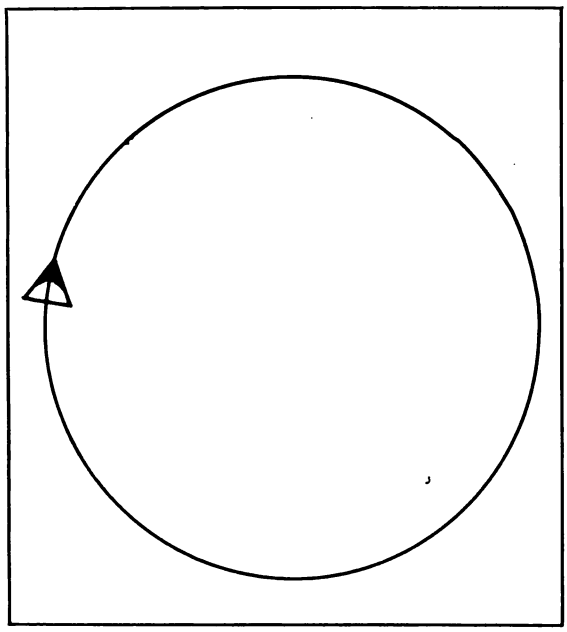
CTRL-G

Thank goodness Logo gives us a way of stopping these infinite loops! If you ever want to stop a procedure at any time, use CTRL-G from the keyboard.

DRAWING MANY CIRCLES

Do you remember that we had Sam draw some circles from before? Let's look at how to draw some more circles. Below is one way to draw a circle. We looked at this procedure in a previous chapter. It does not use the idea of recursion. It simply uses the REPEAT command. Try the procedure again by typing it into the machine.

```
TO CIRCLE
REPEAT 360 [FD 1 RT 1]
END
```



CIRCLE.

Did you notice that there is a special number for Sam in the procedure? The number is 360 of course. As we mentioned before, 360 often gives special results with the turtle. The special result that comes from the special number is that Sam will draw only one complete circle.

Here is another program that we could use to draw the same circle. The program will do exactly the same as before, but it does it in quite a different way. This time the procedure uses recursion.

```
TO CIRCLE2
FD 1
RT 1
CIRCLE2
END
```

The recursive call is in the next to the last line, just before the special word END. After Sam has gone FD 1 and RT 1, he comes to the procedure CIRCLE2, and so he says to himself "I should go and do CIRCLE2." He then stops what he is doing and jumps back to the beginning of the procedure CIRCLE2 again. This means that Sam will be in an infinite loop. If you want to stop the program, you would have to type CTRL-G once more.

Page 149 shows an illustration of Sam on a recursive journey.

Having to stop the procedure with the CTRL-G command can sometimes be a nuisance. There must be some way of stopping a procedure automatically. After all, Sam is really a computer, and computers are supposed to be able to make decisions! Did you know that deciding when to stop is of major importance in computing?

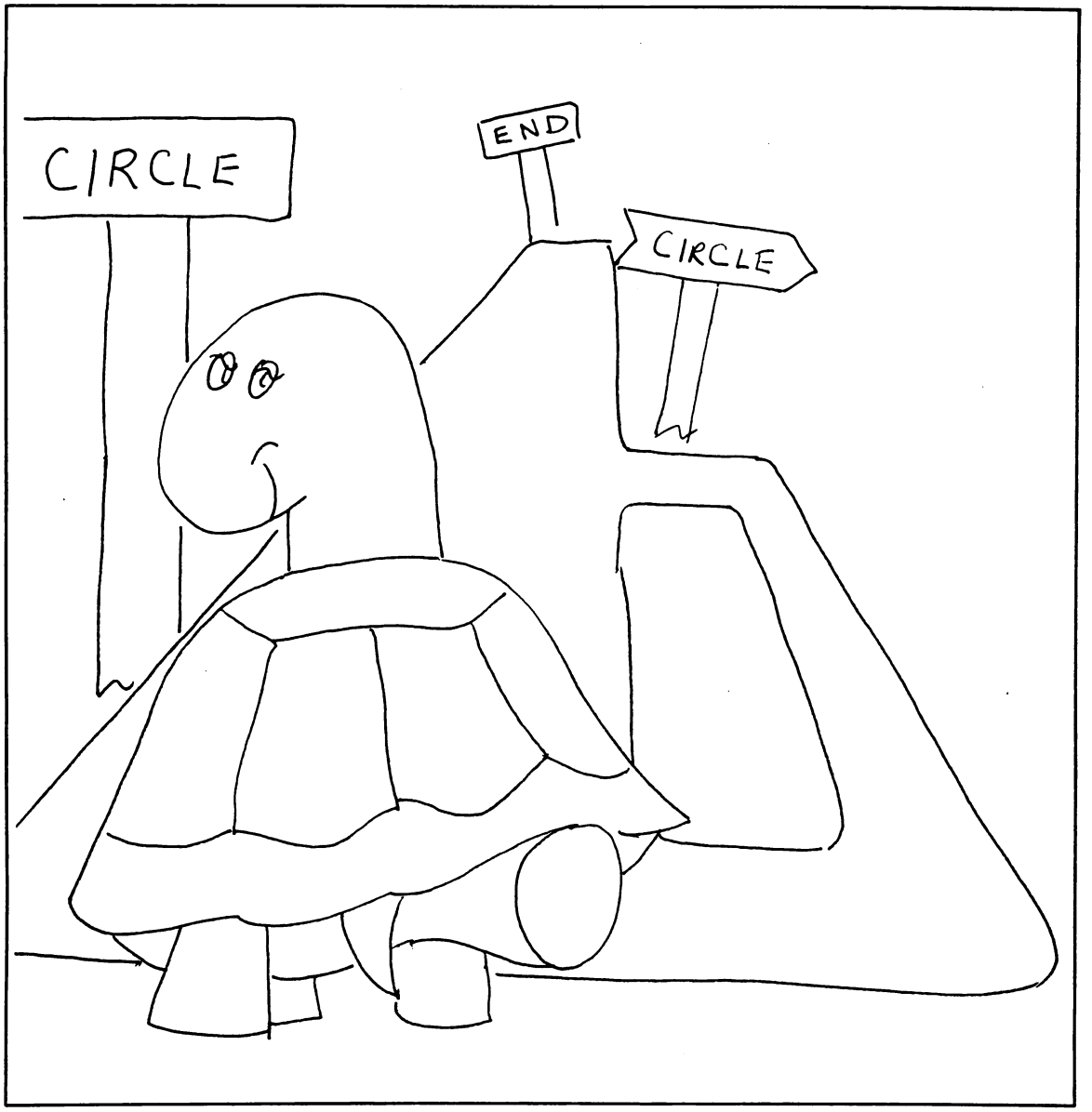
To be able to decide when to stop involves logic. The word IF is the key to this logic, because the word IF can control decisions. For example, you can ask Sam to stop IF he is ready. IF he is back where he started, IF his variables are too large, IF his variables are too small. You could also ask Sam to stop IF he is tired, but of course, let's not forget that Sam is really a computer and computers never get tired!

STOPPING A PROCEDURE IN STYLE

Let's look at the same procedure as before, but this time, let's make Sam stop automatically. Here is a recursive procedure that will stop Sam automatically for us.

```
TO CIRCLE3
FD 1
RT 1
IF HEADING = 0 [STOP]
CIRCLE3
END
```

The first new primitive in our procedure is the IF primitive. The IF primitive allows Sam to make decisions. The other new primi-

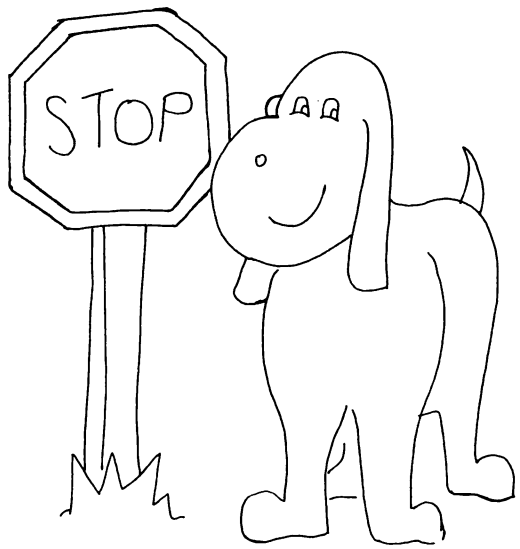


Sam on a recursive path.

tive is the HEADING primitive. HEADING will tell you if Sam is facing in a particular direction. As Sam started off facing north (HEADING = 0), he will have finished the circle when he is again facing north; that is we want Sam to stop when his heading is again equal to zero. STOP is a primitive that will stop our procedure. When Sam meets this primitive, he will quit what he is doing and return to the previous procedure, or *top-level*. If he returns to top-level, he will stop what he is doing and will show you Logo's prompt and cursor.

The primitive IF is very important. Let's

look at it more closely. Here is the IF statement from the above procedure written half in English and half in Logo.



IF the turtle is facing north THEN stop ELSE go to the next primitive in the procedure list. (the next one on *our* list is CIRCLE2.)

The word ELSE is added to help us follow what is happening in the procedure line. You do not need it in the procedure itself. If the turtle is facing north, the procedure will be stopped. If the turtle is not facing north, the procedure will go to the next line to be executed. In this procedure, the next line to be executed is the recursive call to the procedure CIRCLE2.

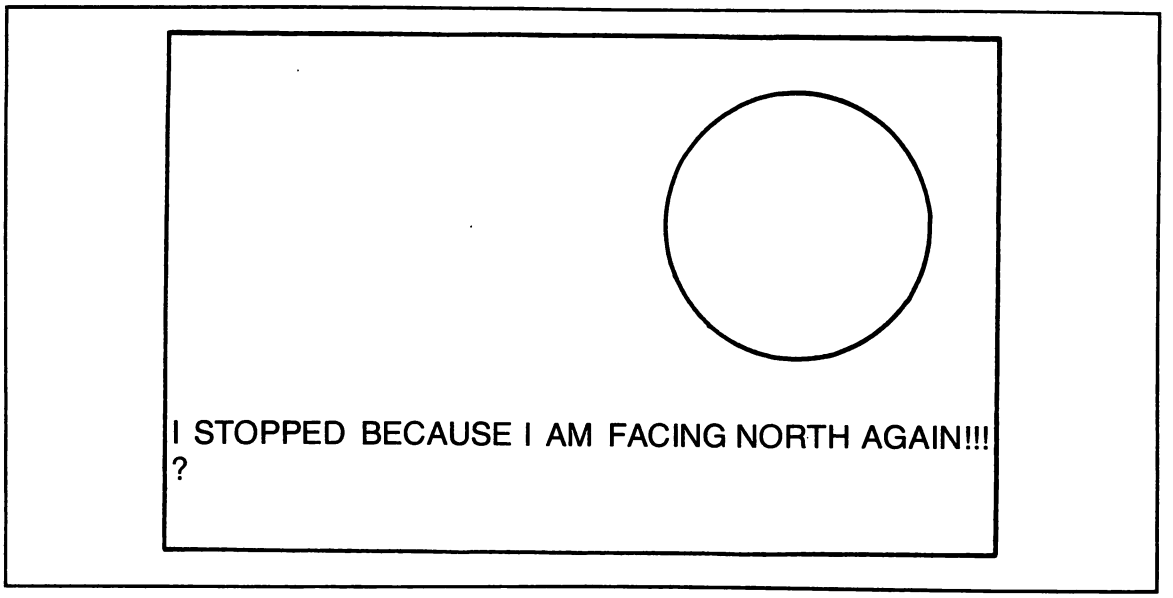
Below is another example and its English/Logo translation. The dots in the procedure indicate that we are simply looking at part of a procedure. The dots represent other primitives that we have left out for the moment. We are only interested in the IF primitive just now.

```
.....
.....
IF :SIZE < 5 [STOP]
FD 200
.....
.....
```

This translates to IF the value in variable SIZE is less than 5 THEN STOP ELSE go forward 200 steps.

```
THAT'S ALL FOLKS!!!
```

Printing a message on the screen.



Circle with a message.

This way of looking at IF is called the IF .. THEN .. ELSE structure. This structure is very important in computing, and it will be used a lot in the following chapters.

Can we use the IF statement to do something else other than simply stopping the program? Let's see in the next section.

PRINTING A MESSAGE ON THE SCREEN

If you prefer, you can have Logo print a message on the screen when the program is stopped after the IF statement. What would be needed is a statement after the IF primitive that will make Sam to send the message to the screen. Sam, of course, would have to make the screen a splitscreen so that there will be a place for him to display his message!

Here is an example of what you might write in a procedure to deliver a message to the screen.

```
...
...
IF :SIZE < 20 [SPLITSCREEN PR
[THATS ALL FOLKS!!!] STOP]
....
...
```

If you use this in one of your procedures, you will see the message shown on page 150 when the variable SIZE is less than 20.

The primitive PR stands for PRint. The message that follows must be in square brackets. The message that you type in the brackets will be sent to the screen.

Here is the circle procedure from before modified to print a message on the screen at the end of the procedure.

```
TO CIRCLE4
FD 1
RT 1
IF HEADING = 0 [SPLITSCREEN
PR[I STOPPED BECAUSE I
AM FACING NORTH AGAIN!!]
STOP]
CIRCLE4
END
```

Type CIRCLE4 to see Sam talk to you.

Can you write a procedure that will print the message THIS PROGRAM HAS BEEN TERMINATED!!! on the screen? Arrange it so that the program will be terminated after Sam has drawn a picture of your choice.

We have wandered away from recursion a little. Let's look at recursion again to see what it really can do for us.

RECURSION AND WHAT IT MEANS TO SAM

Recursion makes Sam a very powerful turtle indeed. In fact, recursion is one of the most powerful operations that a computer can perform. Some languages don't have recursion capabilities built into them, but Sam is lucky, because Logo does have that capability. When we make a recursive call at the end of a procedure, we will get an effect that is a lot like repetition. If the recursive call is not at the end of a procedure then we will find that Sam behaves in unusual ways. This is because a recursive call will create an image of each procedure, complete with its own library of variables. When each recursive procedure is completed, the turtle will wind its way back through the unfinished procedures. This is considered *advanced* Logo, and we will only look at simple recursions here.

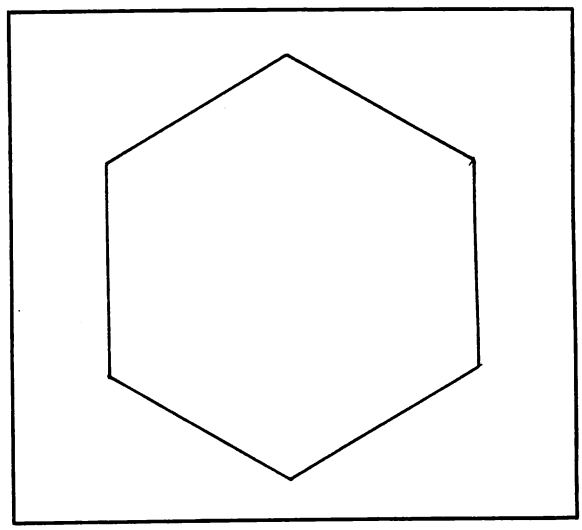
POLYGONS

Here is a group of very well known procedures, which you may have seen before. We will start with a simple procedure, with variables and recursion, that will draw a polygon. After that we will modify our procedure a little to make it do some very different graphics. Finally, we will introduce a third variable that will allow us to control everything that the procedure does.

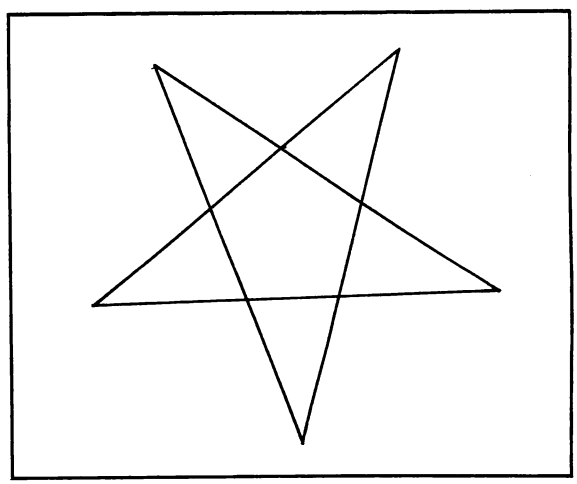
Here is the first procedure.

```
TO POLY1 :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY1 :SIDE :ANGLE
END
```

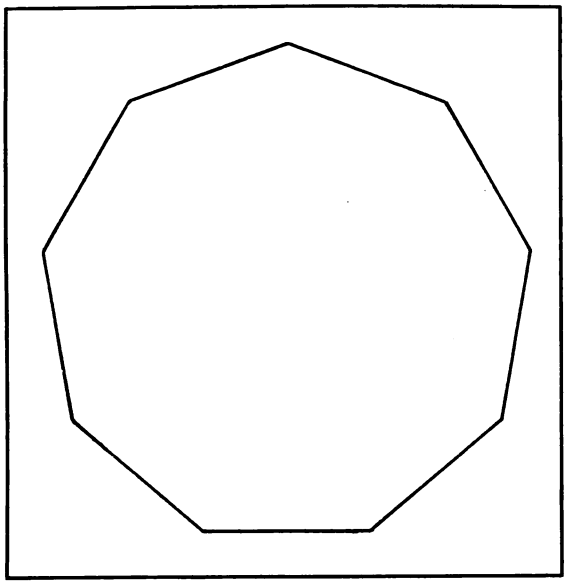
Try typing POLY1 50 60 to begin with, then try POLY1 100 144 and POLY1 30 40. Here is what you will get with each of these values for :SIDE and :ANGLE.



POLY1 50 60.



POLY1 100 144.



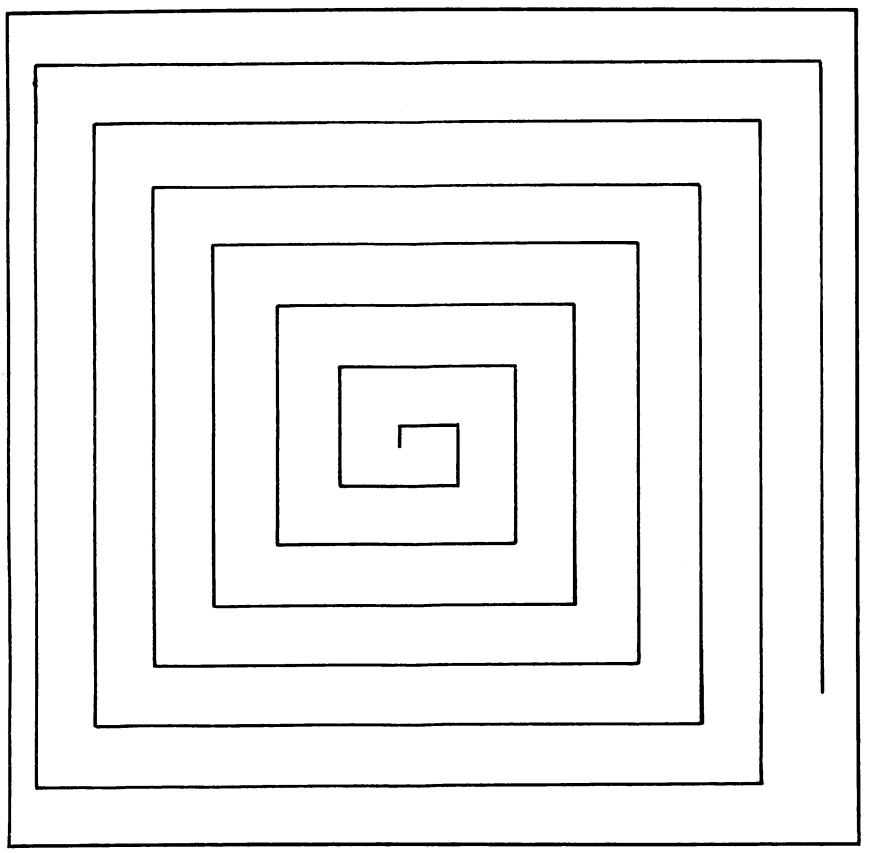
POLY1 30 40.

When you have tried these, try some other values for :SIZE and :ANGLE.

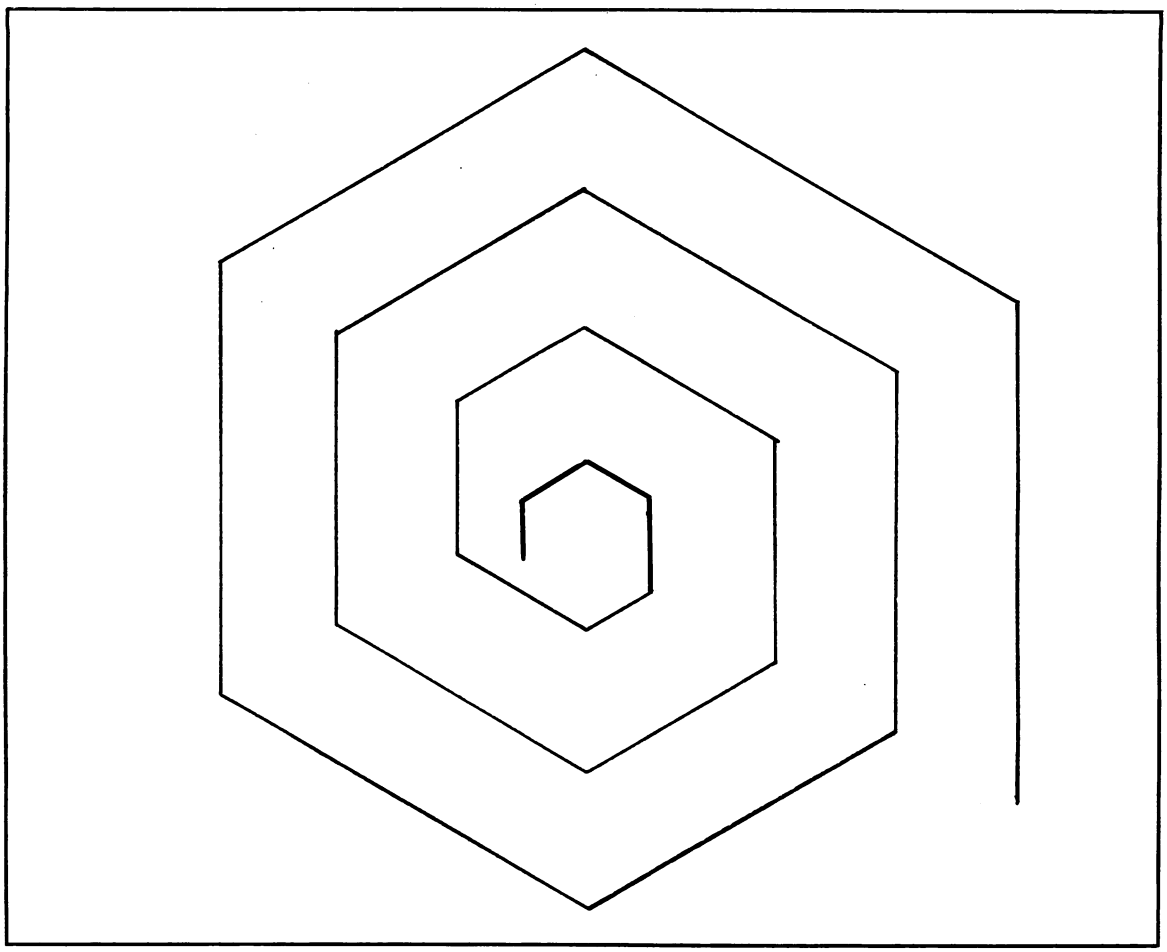
Did you like the diagrams that Sam drew for us? The procedure works by telling Sam to go forward by the amount :SIDE and then to turn to the right by the amount :ANGLE. Can you see the recursive call in the procedure? It comes just before the END statement. The recursive call POLY1 :SIDE :ANGLE also transfers the variables to the next procedure.

Here is a modification of the POLY1 procedure. We will call it POLY2.

```
TO POLY2 :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY2 :SIDE+5 :ANGLE
END
```



POLY2 10 90.



POLY2 10 60.

Try the procedure with POLY2 10 90 to begin with; then try POLY2 10 60. POLY2 30 120 is an interesting one as well!

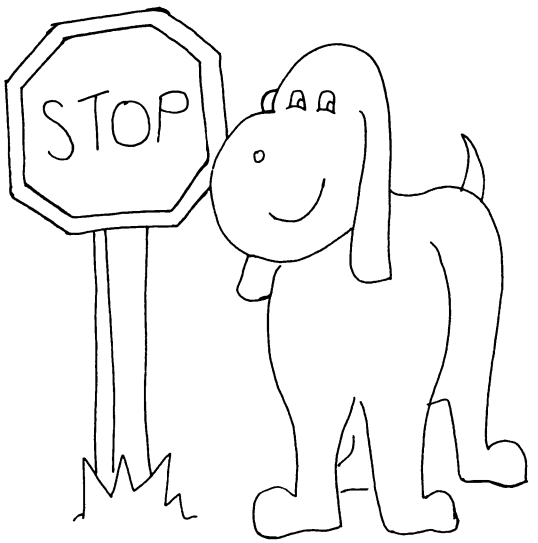
This is a very powerful procedure. You should try all sorts of inputs for the values :SIDE and :ANGLE. The procedure will trace out *spirals*. Can you see which part of the procedure makes it spiral? Correct, the :SIDE+5 is the part that will make Sam spiral. Sam will go forward a little more each time because of the +5 in the :SIDE + 5 primitive.

So far, Sam has gone forward 5 more steps each time. Let's see if we can make that *increment* of five a variable. (An increment is just an increase in a value). We would need to change the procedure to handle a third variable. Here is how we would do that.

```
TO POLY3 :SIDE :ANGLE :INC
FD :SIDE
RT :ANGLE
POLY3 (:SIDE + :INC) :ANGLE
:INC
END
```

Try the procedure with POLY3 10 90 30 to begin with, and then try some other values.

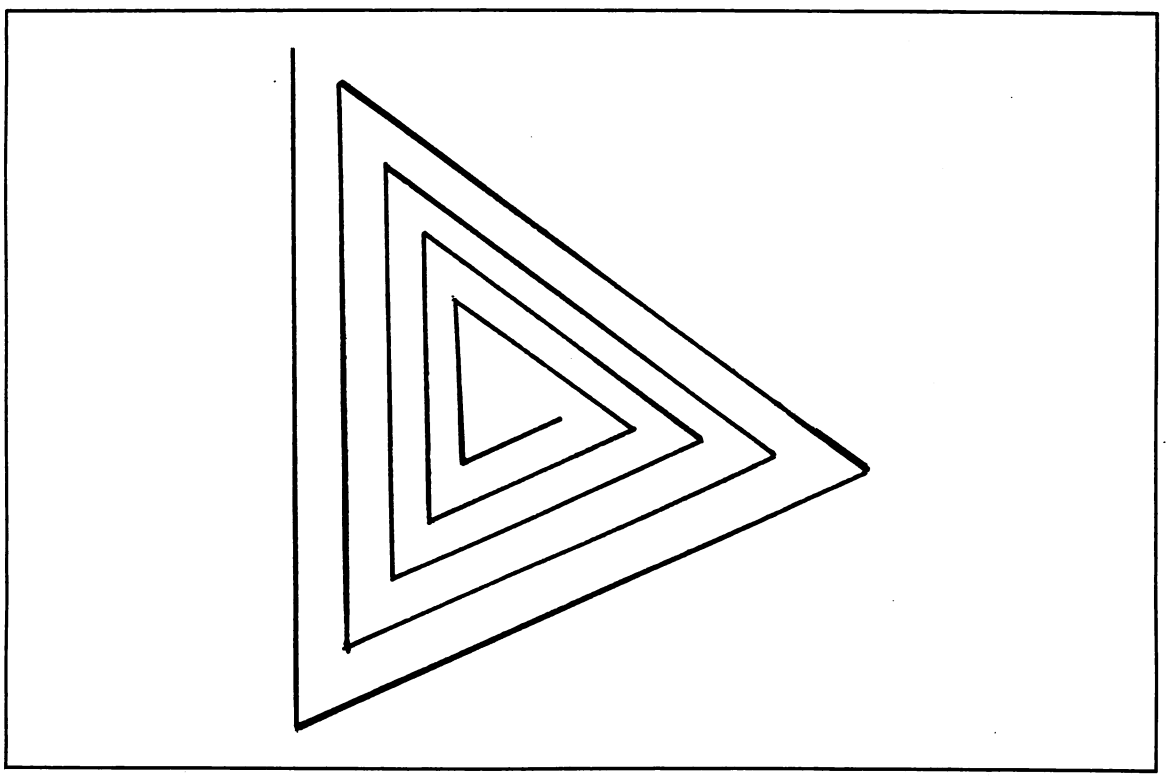
What is different about this procedure? The difference is in the (:SIDE + :INC) statement.



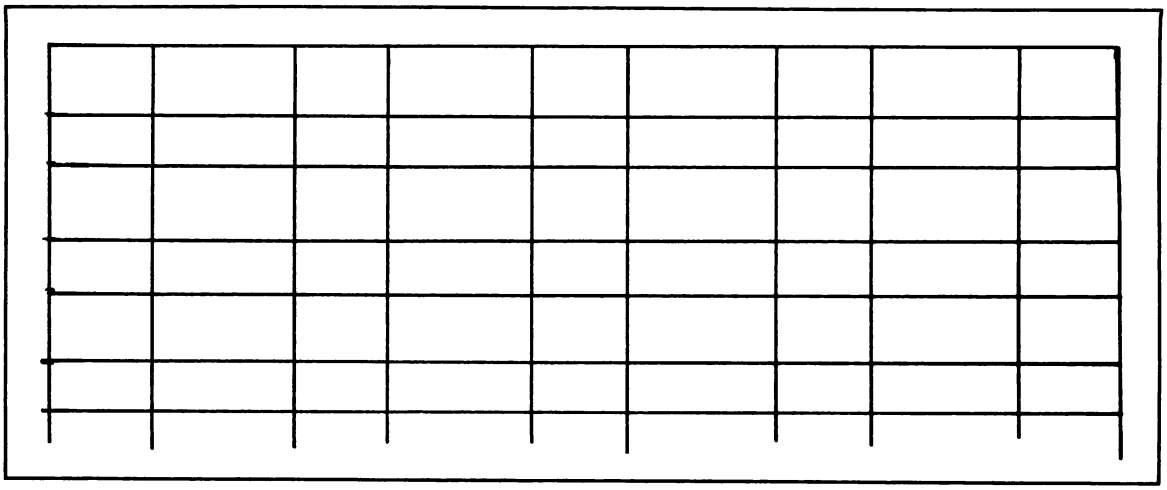
The brackets used here are plain brackets that you type from the keyboard using the SHIFT 8 and SHIFT 9 keys. This type of brackets is used to group things together in Logo. They are different from the square brackets that you get when you use the SHIFT N and SHIFT M keys. In Logo, ( ) and [ ] are used for different purposes.

The brackets used aren't really needed, but they *do* help us to keep track of what is going on. The variable INC, which is short for INCrement, will allow us to control the size of the increment that Sam will add to his forward steps. This increment will be added to the forward step each time the procedure is called.

Have fun with the POLY procedures. Can you think of anything that you might change in the procedures? With the three variables given in the last POLY procedure, you can control



POLY2 130 120.



POLY3 10 90 30.

most of the variables in the drawing of spirals, but there are still plenty of changes that you might make to the procedures that will make significant differences in what they will do.

DRAWING MANY CIRCLES

What would we have to do to draw a lot of circles? If we wanted to draw a lot of circles, we could use recursion along with a simple FD primitive to move Sam to another location. Maybe we should use the PU and PD primitives as well, so that Sam will not draw lines on the screen as he moves to his new location.

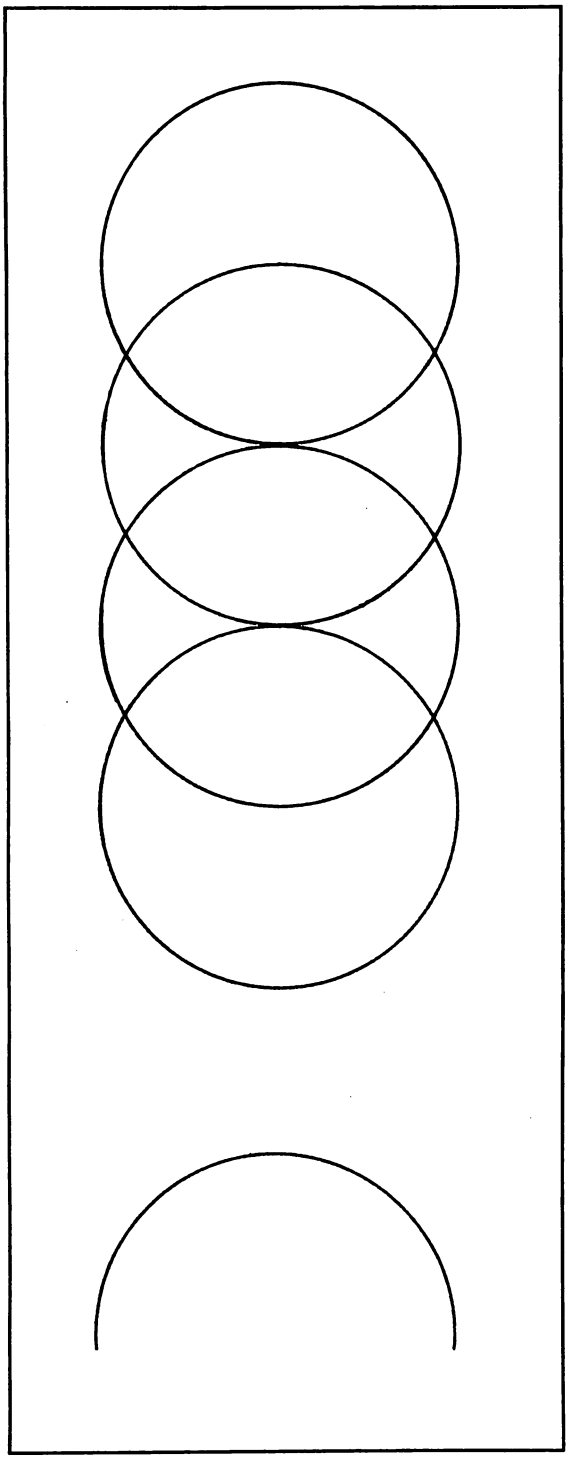
```
TO MANYCIRCLE
CIRCLER 30
MOVE
MANYCIRCLE
END

TO MOVE
PU
FD 30
PD
END
```

Type MANYCIRCLE to see Sam draw the circles. Type CTRL-G to stop the procedure. Can you see the recursive call to the MANYCIRCLE procedure? It occurs just before the END in the MANYCIRCLE procedure.

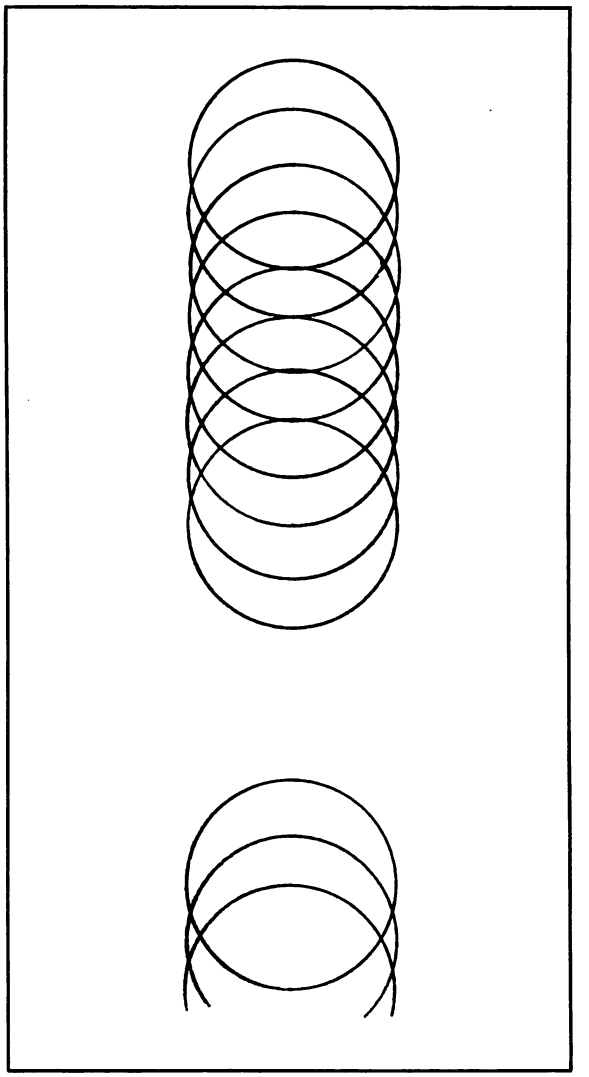
I wonder what we would have to do if we wanted Sam to stop after a certain number of circles; that is, if we told Sam how many circles we wanted him to draw would he be able to draw that many circles and then stop? We would have to supply Sam with the number of circles of course! Now, how could we do that? Hmm. . .what did we look at in the last chapter? Correct, we looked at variables. I wonder if we could use variables and the IF primitive to help Sam figure out when he should stop?

Here is a procedure that will make Sam stop when he has reached the correct number of circles.

```
TO MANYCIRCLE2 :TIMES
CIRCLER 20
MOVE
IF :TIMES=1 [STOP]
MANYCIRCLE2 :TIMES-1
END

TO MOVE
PU
```



MANYCIRCLES.



MANYCIRCLES2 11.

```
FD 10
PD
END
```

To see a given number of circles, type **MANYCIRCLE2 11.** Sam should draw 11 circles for us.

We have used at least three important ideas here. They are recursion, variables, and the IF statement. Did you notice that the vari-

able TIMES is reduced by one each time the procedure is called? The computer word for this action is *decrement*. This is the first time that we have decremented a variable in Logo!

USING THE THINGS THAT WE HAVE LEARNED IN ONE PROCEDURE

Here is a list of some of the things that we have learned so far. Let's look at a challenge that will use all of them.

- Defining a procedure
- Using PU and PD
- Using SETBG and SETPC
- Using recursion
- Using variables
- Changing the value of a variable within a procedure

Here is a challenge: write a procedure that will make Sam draw a lot of rectangles on the screen, with the user being allowed to decide on the length and width of the first rectangle. Each new rectangle will be larger than the one before it.

The user will also be allowed to choose the angle at which the turtle turns before he starts his next rectangle. Use a background color, and a pen color of your choice.

Here is what you want to do in English.

Define the procedure and define the variables length, width and turning angle.
Set the background.
Set the pen color.
Draw the rectangle.
Turn the amount required.
Make a recursive call to the procedure with a new, larger value for length and width.
End

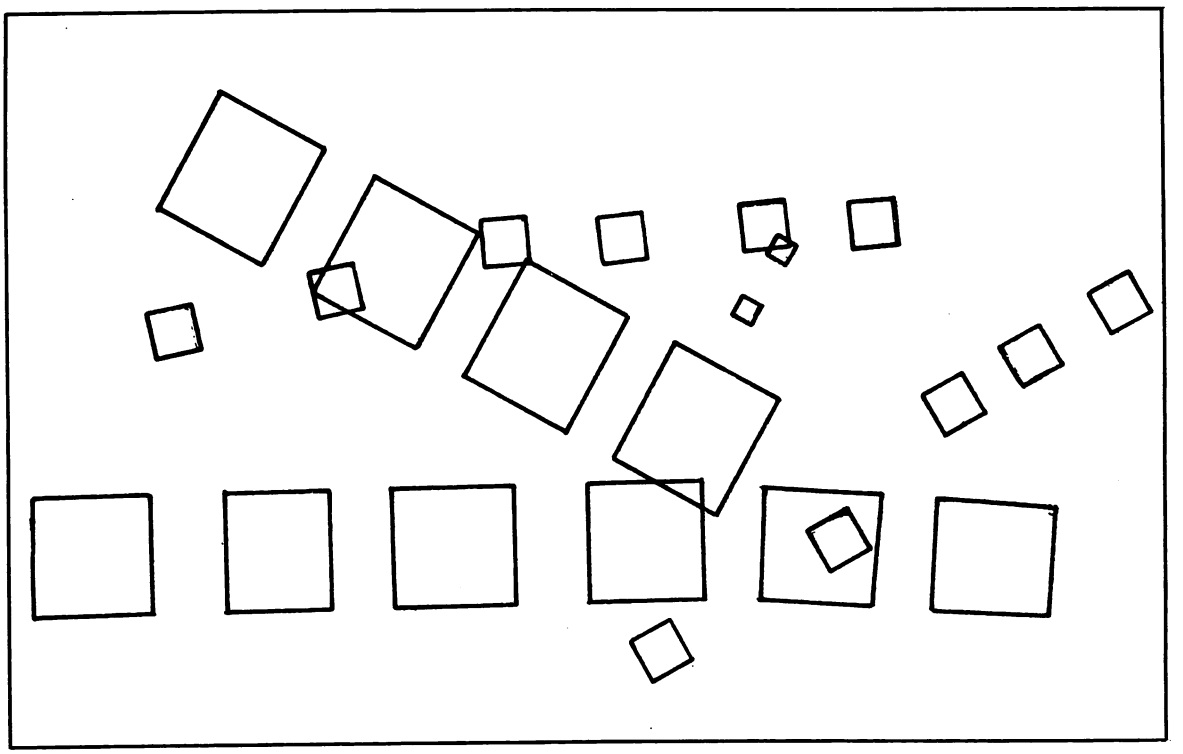
Here is what the translation from English into Logo would look like. Remember that because the procedure is going to include variables, you will have to name them in the TO clause.

```
TO RECTANGLE :LEN :WID
:ANGLE
SETBG 5
SETPC 1
REPEAT 2 [FD :LEN RT 90 FD
:WID RT 90]
PU
RT :ANGLE
FD 40
PD
RECTANGLE :LEN+1 :WID+1
:ANGLE
END
```

Try typing RECTANGLE 1 2 3 to begin with. Notice, that we increased the variables by using the statements :LEN+ and :WID+1. This means that one is added to the contents of the container labeled LEN, and one is added to the contents of the container labeled WID. If LEN has one in it to begin with, it will have two in it after it passes the :LEN+1 statement once and three in it when it passes the :LEN+1 statement the next time, and so on. You should experiment with different inputs. Try RECTANGLE 10 10 90 to see what happens.

The procedure will of course continue until you use CTRL-G to stop it. Can you rewrite the procedure so that it will stop when the length of the rectangle is bigger than 50 turtle steps? Try this for yourself and then look at the next procedure for a solution.

```
TO RECTANGLE :LEN :WID
:ANGLE
IF :LEN>50 [STOP]
SETBG 5
```



RECTANGLE 1 2 3.

```
SETPC 1
REPEAT 2 [FD :LEN RT 90 FD
:WID RT 90]
RT :ANGLE
RECTANGLE :LEN+1 :WID+1
:ANGLE
END
```

Type RECTANGLE 1 2 3 to begin with, and then try other values for the variables.

By the way, did you notice that we could have split the procedure into smaller sub-procedures? Here is what we would have written in Logo if we had decided to do this!

```
TO RECTANGLE :LEN :WID
:ANGLE
IF :LEN>50 [STOP]
SET
DRAW :LEN :WID :ANGLE
MOVE
RECTANGLE :LEN+1 :WID+1
:ANGLE
END

TO SET
SETBG 5
SETPC 1
END

TO DRAW :LEN :WID :ANGLE
REPEAT 2 [FD :LEN RT 90 FD
:WID RT 90]
RT :ANGLE
END

TO MOVE
PU
RT :ANGLE
FD 40
```

```
PD
END
```

The procedure will do the same as before of course! Did you notice that we transferred variables to the DRAW procedure, so that DRAW would be able to use :LEN, :WID and :ANGLE?

Let's take Sam for a spin in the next section.

SPINNING CIRCLES WITH SAM

Here is a procedure to spin a circle. While the circle is spinning, we will tell Sam to make the circle grow bigger. This one should give us an interesting pattern!

The main procedure will look like this.

```
TO SPIN.CIRCLE :SIZE
SET
SPIN :SIZE
END
```

The SET procedure will send Sam to a suitable part of the screen so that he will have room to do his stuff! The SPIN procedure will do the actual drawing of the circles as well as the rotation of each circle.

Here is the SET procedure.

```
TO SET
WINDOW
FULLSCREEN
SETBG 1
SETPC 5
PU
SETPOS [40 80]
PD
END
```

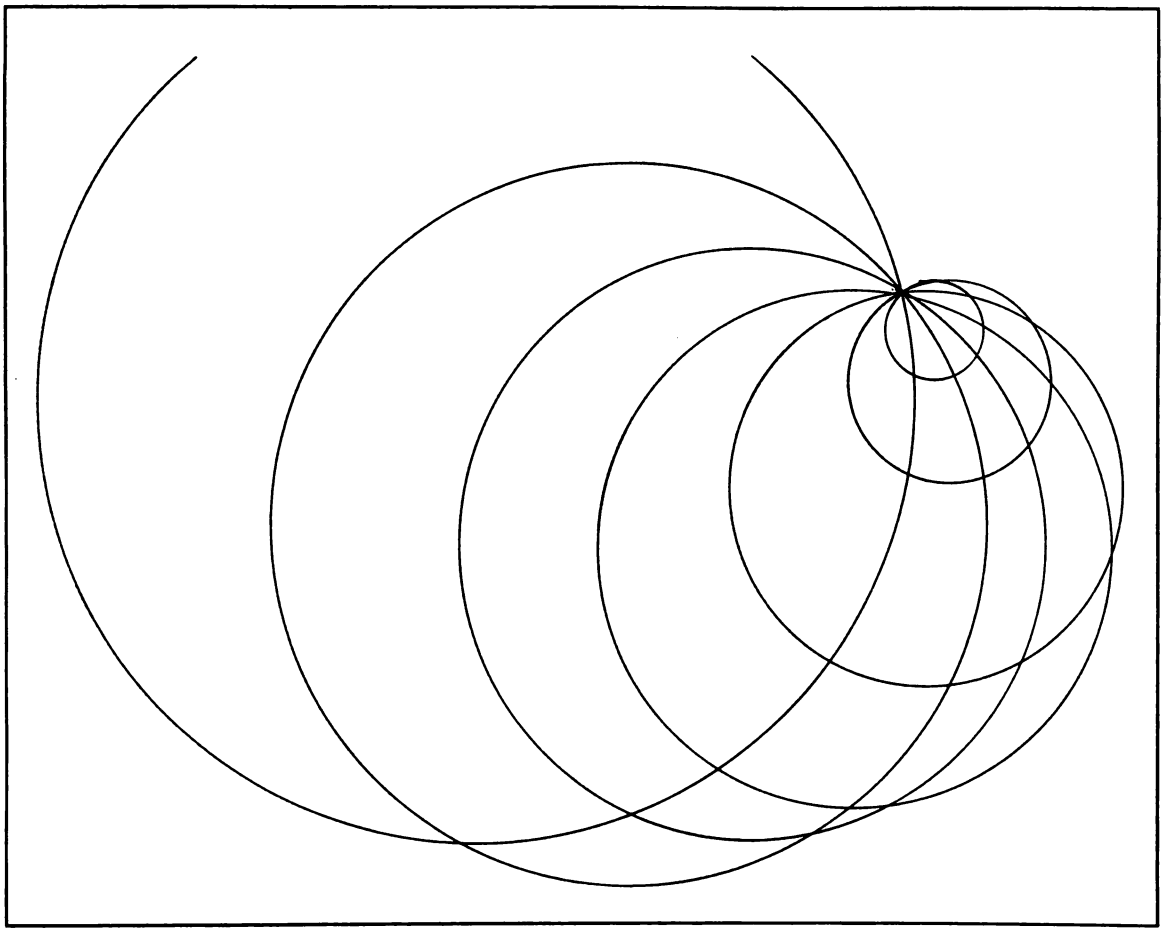
The SETPOS primitive will place Sam at location 40, 80. This seems to be a good starting point for this procedure. This starting point was decided on by trial and error. Try changing the values to see which starting point you prefer.

Here is the heart of this procedure, the SPIN procedure. This procedure uses both variables and recursion.

```
TO SPIN :SIZE
IF :SIZE >100 [STOP]
CIRCLER :SIZE
RT 20
SPIN :SIZE + 10
END
```

Here is the complete procedure. This is what you would have to type if you wanted to see Sam draw the "spinning circle". Remember to use ERPS if you have something in Sam's workspace that you do not need.

```
TO SPIN.CIRCLE :SIZE
SET
SPIN :SIZE
END

TO SET
WINDOW
FULLSCREEN
SETBG 1
SETPC 5
PU
SETPOS [40 80)
PD
END

TO SPIN :SIZE
IF :SIZE >100 [STOP]
CIRCLER :SIZE
RT 20
SPIN :SIZE + 10
END
```



SPINCIRCLE 10.

Type SPIN.CIRCLE 10 to see what happens. Then try SPIN.CIRCLE 30 to look at some larger circles. This drawing looks a little like a seashell. Maybe we should rename and modify the procedure to make it a proper seashell.

MAKING SMALLER AND SMALLER CIRCLES WITH RECURSION

Here is a procedure that uses the CIRCLER primitive that Sam already knows. The idea is to draw smaller and smaller circles on the screen. You get to tell Sam how big the first one is to be! When the circles get too small, Sam will stop the procedure.

Here is our English list.

> Draw a circle, unless it is very small.
> Reduce the radius of the circle by ten units.
> Draw another circle.

Here is the main procedure.

```
TO DRAW.CIRCLES :SIZE
SET
CIRCLE :SIZE
END
```

The SET procedure will send Sam to the lower left hand corner of the screen. Here is what the SET procedure will look like.

```
TO SET
FULLSCREEN
PU
SETPOS [-120 -30]
PD
END
```

The CIRCLE procedure is the procedure that will draw the first circle and then reduce the radius by 10 so that it can draw the next circle, and so on. Here is the CIRCLE procedure.

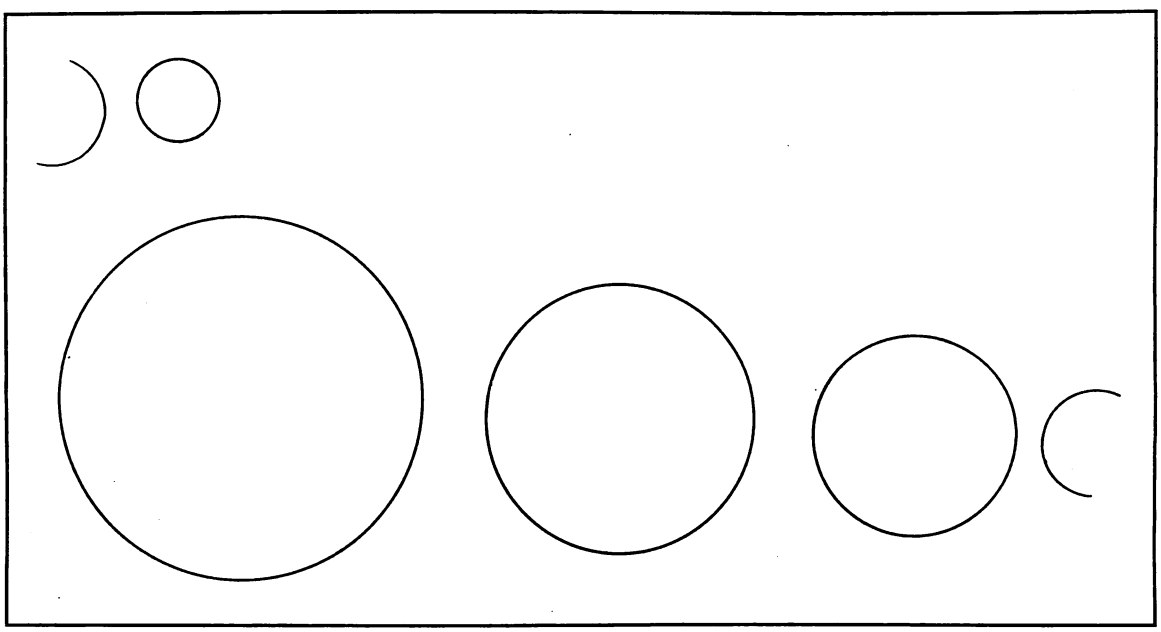
```
TO CIRCLE :SIZE
IF :SIZE < 1 [STOP]
CIRCLER :SIZE
MOVE :SIZE
CIRCLE :SIZE - 10
END
```

The IF statement will stop the program if the circles become too small. Sam doesn't look good drawing very small circles, and circles with negative radius are unusual! The MOVE procedure will simply move Sam over and up a bit so that he can draw his next circle. Here is the MOVE procedure

```
TO MOVE :SIZE
PU
RT 90
FD :SIZE * 2.0
LT 90
FD :SIZE * 0.5
PD END
```

The FD :SIZE *2.0 will move Sam over the radius of his last circle times two. This should bring him to just about the correct position to make a presentable picture.

After the MOVE procedure is carried out, Sam will obey our recursive call to CIRCLE :SIZE - 10 once more—good old obedient Sam!



DRAW.CIRCLE 50.

Here is the complete procedure. This is what you would have to type in to see Sam draw those circles. Remember to use ERPS if it is necessary.

```
TO DRAW.CIRCLES :SIZE
SET
CIRCLE :SIZE
END

TO SET
FULLSCREEN
PU
SETPOS [-120 -30]
PD
END

TO CIRCLE :SIZE
IF :SIZE < 1 [STOP]
CIRCLER :SIZE
MOVE :SIZE
CIRCLE :SIZE - 10
END

TO MOVE :SIZE
PU
RT 90
FD :SIZE * 2.0
LT 90
FD :SIZE * 0.5
PD END
```

Type DRAW.CIRCLE 50 to begin with, and then try some other values for the radius.

SAM'S TOWER REVISITED

Do you remember that we had Sam draw a tower of blocks in the last chapter? We taught Sam to do this without the use of recursion. Recursion, as we said before, is very powerful. Let's look at how to draw a similar tower, but this time, let's see if we can do it using recursion.

Here is a list of English statements telling what we want Sam to do.

Set Sam's position on the screen to a suitable location.
Draw smaller and smaller blocks on the screen until they become too small.
End.

Our list seems fairly short this time. Let's see what the procedures look like.

Here is the main procedure.

```
TO TOWER :SIZE
SET
DRAW.BLOCKS :SIZE
END
```

The SET procedure is similar to the others. We could write a list of English instructions for it as well, but we can also simply use the words in our procedure instead. It turns out that they are almost the same, thanks to the way Logo is designed.

Here is the SET procedure.

```
TO SET
FULLSCREEN
SETBG 2
PU
SETPOS [-30 -110]
PD
END
```

The next procedure that we should look at is the DRAW.BLOCKS procedure. Here is the procedure.

```
TO DRAW.BLOCKS :SIZE
IF :SIZE < 5 [STOP]
SQUARE :SIZE
MOVE : SIZE
```

```
DRAW.BLOCKS :SIZE – 10
END
```

This procedure is not quite so easy. It is the procedure that contains the recursive call to itself! Let's look at the list of English instructions for the procedure.

Draw the blocks over and over until the next one is less than 5 units in length.
After drawing a block, MOVE over and up to get ready for the next block.
Draw another block, but this time reduce its size by ten units.
End

Did you notice that there is a SQUARE :SIZE procedure in the middle of the DRAW.BLOCKS procedure? This is the procedure that will actually draw the squares. The value of SIZE will be transferred to the procedure so that the square will be the correct size for the tower.

Here is the SQUARE :SIZE procedure.

```
TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END
```

The procedure is very ordinary, Sam almost knows this one by heart!

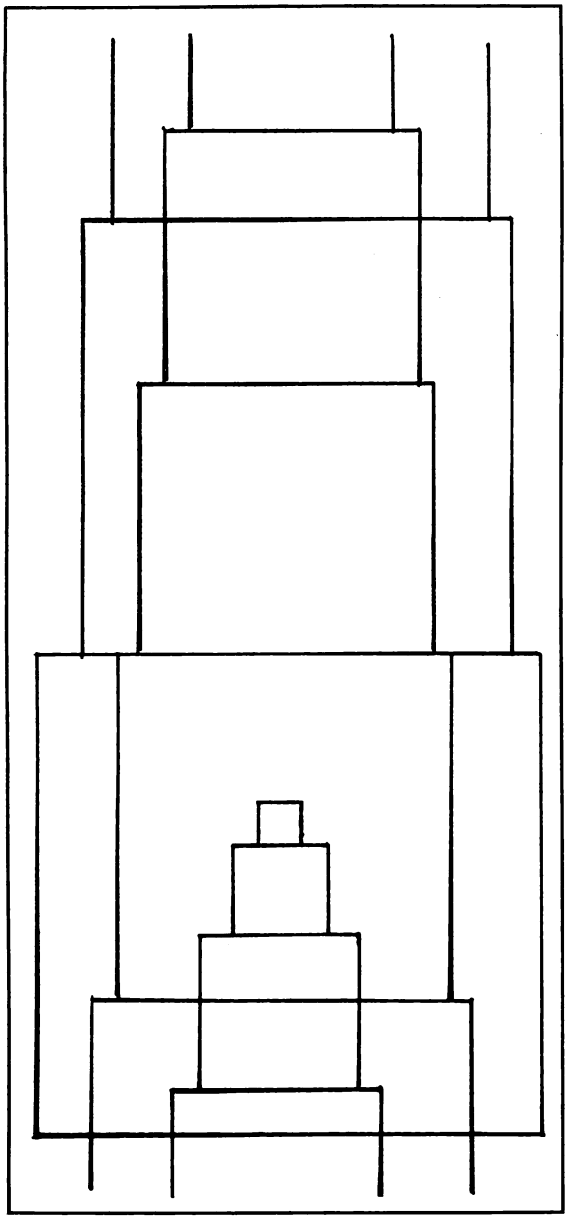
The MOVE procedure is the next one that we should look at. It, like most MOVE procedures, is very exact, and it must do its job in just the right way. Here is the procedure that we need.

```
TO MOVE :SIZE
FD :SIZE
RT 90
FD 5
LT 90
END
```

Why do you think that we told Sam to go forward by the amount :SIZE? The value of SIZE will be the length of the block that Sam will have to move around. The RT 90 will turn the corner, and the FD 5 will take Sam forward on the top edge of the cube. The difference in the dimensions of the boxes is 10 units, and the FD 5 is half of that amount! Finally, to get Sam ready for the next block, you will need to make him turn LT 90. If that seems difficult, try playing turtle!

Here is the complete procedure. Type this in to see how Sam will draw his tower.

```
TO TOWER :SIZE
SET
DRAW.BLOCKS :SIZE
END

TO SET
FULLSCREEN
SETBG 2
PU
SETPOS [–30 –110]
PD
END

TO DRAW.BLOCKS :SIZE
IF :SIZE < 5 [STOP]
SQUARE :SIZE
MOVE : SIZE
DRAW.BLOCKS :SIZE – 10
END

TO SQUARE :SIZE
REPEAT 4 [FD :SIZE RT 90]
END

TO MOVE :SIZE
FD :SIZE
```



TOWER 100.

```
RT 90
FD 5
LT 90
END
```

That must be one of the longest procedures so far! Type TOWER 100 to see Sam draw the tower. Good old Sam—he always ends up on the very top story of the tower! Did you notice that he also ended up on the top of the pyramid? I guess robots and turtles are just like people; they like to climb to the top of things! Can you write a procedure that will draw this tower at an angle? How about allowing the user to specify the angle?

SAM'S PYRAMID REVISITED

Do you recall that we taught Sam how to draw a pyramid? The pyramid was drawn by telling Sam to draw TRI 100, then TRI 80, then TRI 60, and so on. Let's see if we can use recursion to do the same thing. Recursion should make a shorter more elegant procedure. Let's see if it does!

Here is what we want to do set in a list of English statements.

> Set the screen, and move Sam to his starting point.
> Tell Sam to stop if the triangles get too small.
> Tell Sam to draw his first triangle.
> Tell Sam to draw another triangle using the same procedure, but this time, reduce the length of the triangle's side.
> End.

Here is the main procedure that will call the subprocedures.

```
TO PYRAMID :SIZE
SET
DRAW.TRI :SIZE
END
```

The SET procedure is very similar to others that we have used, here is what it looks like.

```
TO SET
FULLSCREEN
SETBG 3
PU
SETPOS [0 100]
RT 150
PD
END
```

The SETPOS [0 100] will move Sam to the middle and top of the screen. The RT 150 will turn him so that he will draw the first triangle with its base parallel to the bottom of the screen.

Here is the DRAW.TRI procedure that was listed in the control procedure.

```
TO DRAW.TRI :SIZE
IF :SIZE < 5 [SETHEADING 0
STOP]
TRI :SIZE
DRAW.TRI :SIZE - 10
END
```

Can you see the IF primitive? This will STOP Sam in style! Why do you think that we told Sam to SETHEADING 0? Maybe you will be able to answer this question after you have a chance to run the program!

The recursive call occurs just before the END statement. This call will send Sam back to the beginning of the procedure DRAW.TRI again. This time, Sam will return to the procedure with the value of SIZE reduced by ten units! If you wanted, you could change the DRAW.TRI :SIZE-10 to DRAW.TRI :SIZE-20 to see what would happen.

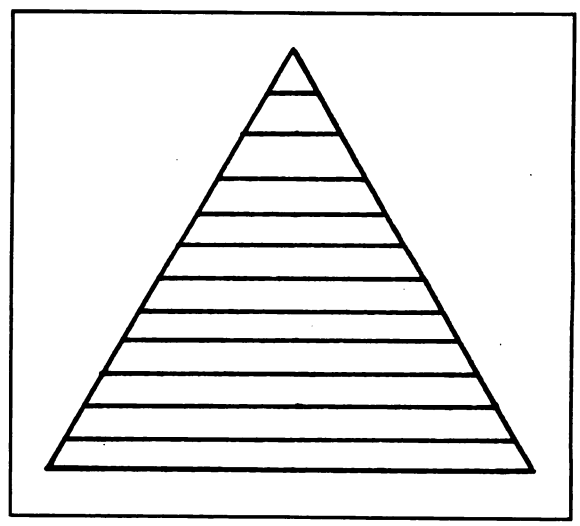
Here is the TRI :SIZE procedure itself.

```
TO TRI :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

I guess that the triangle procedure will never change!

Here is the complete procedure. Type all of this in to see Sam draw his pyramid using recursion.

```
TO PYRAMID :SIZE
SET
DRAW.TRI :SIZE
END

TO SET
FULLSCREEN
SETBG 3
PU
SETPOS [0 100]
RT 150
PD
END

TO DRAW.TRI :SIZE
IF :SIZE < 5 [SETHEADING 0
STOP]
TRI :SIZE
DRAW.TRI :SIZE - 10
END
```



PYRAMID 100.

```
TO TRI :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

Type in PYRAMID 200 to see Sam draw a nice large pyramid. Then try for a smaller version!

From Egyptian pyramids to flowers, is there anything that Sam can not do?

USING RECURSION TO MAKE A DAISY CHAIN

Let's use the FLOWER procedure in our next procedure. What could we do with a procedure called FLOWER? We could draw a bunch of flowers, or we could draw a garden with flowers in it, or we could link some flowers to draw a daisy chain.

The daisy chain idea might be good! What would we have to do to link some of the flowers together? Here is an outline in English.

Set the screen mode to WINDOW so that our picture doesn't overdraw itself.
Set Sam over to the lower right of the screen somewhere.
Draw the daisy chain using the FLOWER procedure.
End.

Here is the main procedure for the daisy chain.

```
TO DAISY.CHAIN :SIZE
SET
CHAIN :SIZE
END
```

This is quite a short procedure, but I think you will be surprised at what it will do in the end! The subprocedure CHAIN :SIZE is the procedure that will call the procedure FLOWER :SIZE. SET will simply set the screen mode and move Sam to the right starting point on the screen. The CHAIN procedure has a variable attached to it. This will allow you to decide on the size of chain and the size of flower that Sam draws.

Here is the procedure for SET.

```
TO SET
WINDOW
FULLSCREEN
SETBG 5
PU
SETPOS [-100 -10]
SETHEADING 0
PD
END
```

Can you remember what color SETBG 5 will give you? Of course, it will give you blue. Maybe we should have used green for our daisy chain background. You can change that later if you like. What do you think SETHEADING 0 will do? It will tell Sam to face north. This will set him off in the right direction to begin with.

Let's draw the chain now. Here is the procedure that you will need.

```
TO CHAIN :SIZE
REPEAT 6 [FLOWER :SIZE]
END
```

Aha! There is the FLOWER procedure at last!

Here is the FLOWER procedure, or at least the control procedure for FLOWER. There is something a little different from the last FLOWER procedure. Can you see what it is?

```
TO FLOWER :SIZE
STALK :SIZE
HEAD :SIZE
END
```

Of course, we need to make sure that Sam already knows how to STALK, and HEAD and of course PETAL, which is used in the HEAD procedure. The procedures for STALK, HEAD, and PETAL will be the same as before, so we will not look at these very closely.

Do you recall that you can run your procedures as you go along? You do not have to wait until you have finished a complete procedure before you try out the subprocedures. For instance, you can type FLOWER and see what the flower looks like without waiting for the complete daisy chain procedure to be typed in. This feature of Logo helps to debug programs!

Here is the entire procedure for drawing our daisy chain.

```
TO DAISY.CHAIN :SIZE
SET
CHAIN :SIZE
END

TO SET
WINDOW
FULLSCREEN
SETBG 5
PU
SETPOS [-100 -10]
SETHEADING 0
PD
END

TO CHAIN :SIZE
REPEAT 7 [FLOWER :SIZE]
END

TO FLOWER :SIZE
STALK :SIZE
HEAD :SIZE
END

TO STALK :SIZE
ARCR :SIZE 60
END

TO HEAD :SIZE
REPEAT 9 [PETAL :SIZE RT 60]
END

TO PETAL :SIZE
ARCR :SIZE*0.25 80
RT 100
ARCR :SIZE*0.25 80
END
```

To see Sam draw the daisy chain, type DAISY. CHAIN 100. Then you can try typing DAISY:CHAIN 200, to see the difference between using 100 and 200.

Were you surprised that the chain came right back and joined up with itself? I wonder why Sam did that? There must have been a combination of numbers that came to 360 somewhere. Can you see where the 360 comes from in the procedure?

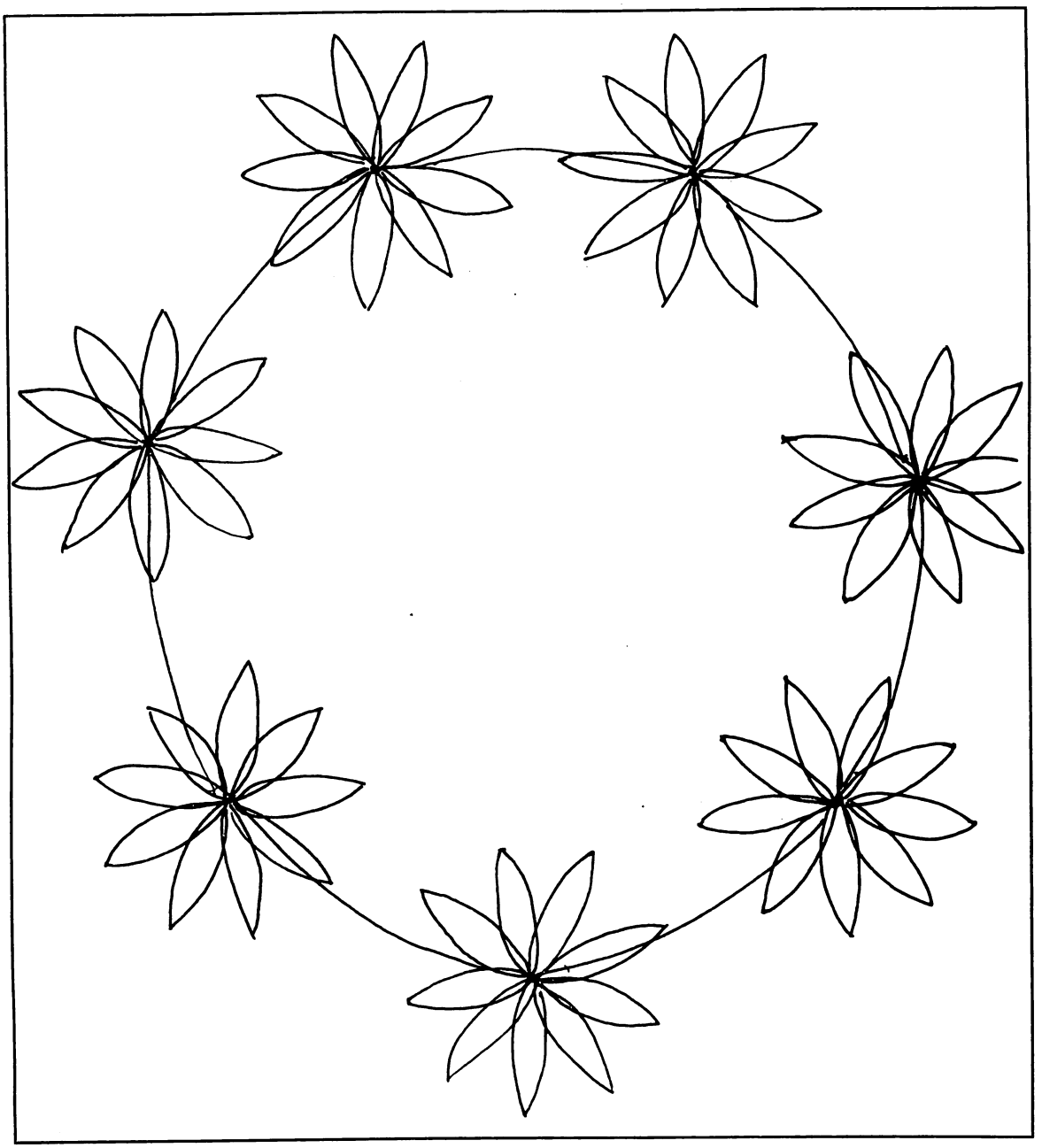
SUMMARY

Here is a summary of the new things in Logo that we have covered in this chapter.

1. Recursion means that you can tell Sam to tell himself to do something.
2. Recursion is a powerful idea, and you should use it often in your procedures.
3. A simple example of recursion would be

```
TO CIRCLE
FD 1
RT 1
CIRCLE
END
```

   The call to CIRCLE just before the END statement is the recursive call in the procedure.
4. If you are using recursion, you may have to stop the procedure using CRTL-G, or you can stop the procedure by using an IF



DAISY.CHAIN 100.

statement and by giving the turtle a suitable limit.

5. You can print messages on the screen by using the PR primitive. If you use

```
PR [HELLO THERE]
```

in a procedure, you will get the message HELLO THERE printed on the screen.

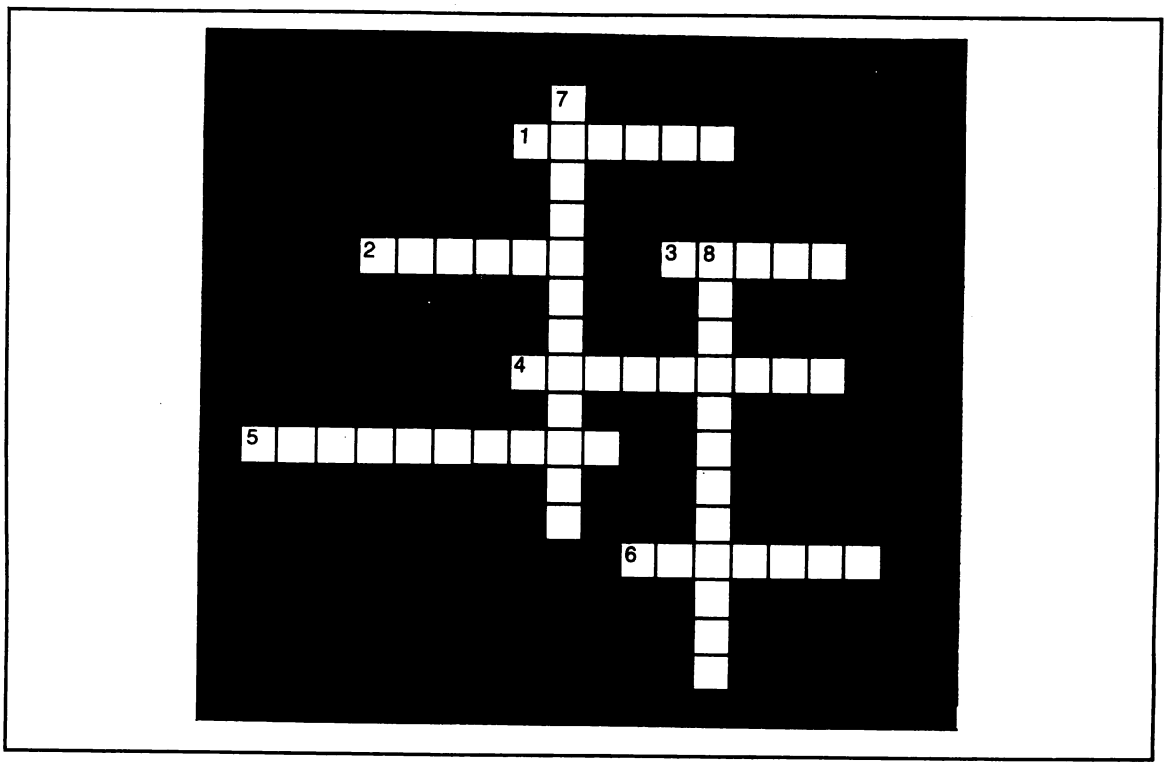
CHALLENGES

1. Write a procedure to draw a pattern of circles using recursion.
2. Write a procedure to draw a pattern of circles using recursion. Allow the user to specify the values used to determine the size of the circles.
3. Write a procedure to draw a pattern of even larger circles using recursion. Allow the user to specify the values used to determine the size of the circles.
4. Write a procedure to draw a pattern of arcs using recursion.
5. Write a procedure to draw a pattern of arcs using recursion. Allow the user to specify the values used to determine the size of the arcs as well as the number of degrees of arc. Use recursion to draw ever larger arcs.
6. Write a procedure to draw a pattern of arcs using recursion. Allow the user to specify the values used to determine the size of the arcs as well as the number of degrees of arc. Use recursion to draw ever-larger arcs. Allow the user to control the position of the turtle as he moves between recursions.

QUIZ

1. Recursion means that you can tell Sam to tell _____ to do something.
2. Recursion is a _____ idea, and you should use it in your procedures often.
3. A simple example of recursion would be

```
TO CIRCLE
FD 1
RT 1
_____
END
```



The call to _____ just before the END statement is the _____ call.

4. If you are using recursion you may have to stop the procedure using CTRL-____, or you can stop the procedure by using an _____ and by giving the turtle a suitable limit.
5. You can print messages on the screen by using the _____ primitive. If you use

```
PR [HELLO THERE]
```

then you will get the message _____ printed on the screen.

CROSSWORD PUZZLE

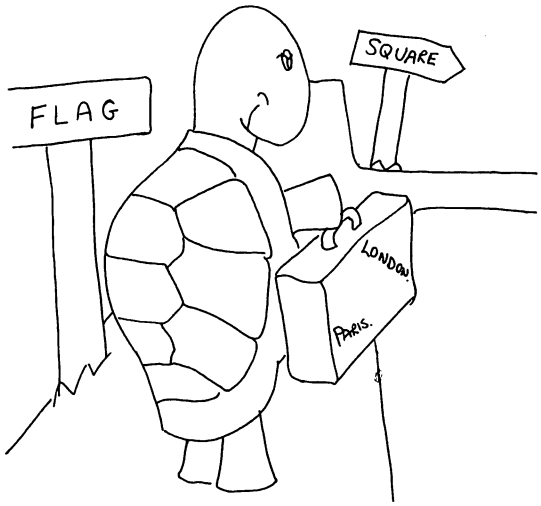
Clues Across

1. To continue after a pause, you press _____.
2. You could use this to decide when Sam will stop.
3. This will stop a procedure from the keyboard.
4. Telling Sam to tell Sam is called _____.
5. Doing things over and over again is called _____.
6. This will give a pause.

Clues Down

7. When Sam does something forever and forever, he is in an _____.
8. Sam does this if you use recursion.

Chapter 9



Drawing Pictures with Sam

We know quite a lot about Logo and its primitives now. Let's look at how to use these ideas to draw some pictures with Sam.

SOME THINGS WE KNOW THAT WILL HELP

Here is a list of things that will be useful in this chapter. The list is a summary of the things we have looked at from Chapter 1 to Chapter 8.

1. We can tell Sam to go forward, backward, right, and left by using the commands FD, BK, RT, and LT. We must supply a space followed by a number for each of these commands. For example, BK 20 will make Sam go back 20 steps.

2. We can teach Sam the meaning of a word by defining the procedure with the word TO. For example, if we wanted to teach Sam the meaning of something called "SQUARE" then we would type.

```
TO SQUARE
...
...
END
```

We always close a procedure with the special word END.

3. We can tell Sam to repeat things over and over again. This is done using the REPEAT primitive. For example, REPEAT 5 [FD 10 RT 30] would repeat the primitives FD 10 and RT 30 five times.

4. We can play at being Sam. If you are unsure of what you should tell Sam to do next, you can pretend that you are Sam and walk around the room just as Sam walks around the screen. When you come to the part where you are stuck, you can ask yourself "What should Sam do next?".

5. If you want to make Sam do something complicated, break the problem down

into smaller pieces. Make each piece a separate procedure. Build all of these smaller procedures into a larger procedure. For example, a house is made up of walls and a roof. We would need three procedures to draw the house. The first would be called HOUSE; the second would be called WALL; and the third would be called ROOF.

6. We can use lists of English statements to help us with solving problems, especially with breaking large problems into smaller ones.

7. We can use the editor to make changes to our procedures. This saves us a lot of typing and allows us to experiment with the changes that we want to make to the procedure. To enter the editor and make changes to the procedure SQUARE, we would type ED "SQUARE. When we are in edit mode we can use the editor CTRL keys to make the changes that we want to the procedure. Look back to Chapter 4 for a list of the possible edit commands.

8. We can use different colors for Sam's background and for the color of Sam's pen. For example, SETBG 5 would set the background color to blue. SETPC 2 would set the pencolor to green. We can also lift the pen using PU and lower the pen using PD. Look back to chapter 5 for a list of colors and their code numbers.

9. We can change Sam's screen area by using the commands WINDOW, FENCE, FULLSCREEN, and SPLITSCREEN. If you want to change the screen from the keyboard, you can do so by using the control keys. For example, CTRL-L will give you the complete screen for Sam to play on. Look back to Chapter 5 for the details.

10. We can tell Sam to draw circles and arcs. If you want to draw a circle with a radius of 50, you would type CIRCLER 50 for a circle to the right. You would use CIRCLEL for a circle to the left. If you want to draw a left arc with a radius of 40, and 60 degrees of arc, you would tell Sam to ARCL 40 60. For an arc to the right, you would use ARCR 40 60.

11. You can tell Sam to make an object large or small by using variables. Variables are given names that help you to remember what they represent. If you want to use a procedure with a variable that represents the length of the side of a square, you might define that procedure by typing TO SQUARE :SIDE. We pronounce :SIDE as "dots SIDE".

We can also change the size of a variable within a procedure by using primitives like

```
...
...
SQUARE :SIZE+3
...
...
```

The :SIZE+3 will increase the variable SIZE by adding three to its last value. If :SIZE has a value of 6, then :SIZE+3 will give a new value of 9 for :SIZE.

12. You can *call* a procedure from inside another procedure. By using this idea, you can break your problem into smaller parts.

13. If a procedure calls itself, then you are using *recursion*. Recursion often gives you an infinite loop. You can stop the procedure by typing CTRL-G.

14. You can use the IF statement to control the computer. The IF statement can be used to tell when the computer should stop executing a procedure. The IF statement is often used with recursion so that a procedure will stop on a given condition, for example

```
IF :SIZE > 200 [STOP]
```

PLANNING YOUR PICTURE OR MAKING IT UP AS YOU GO

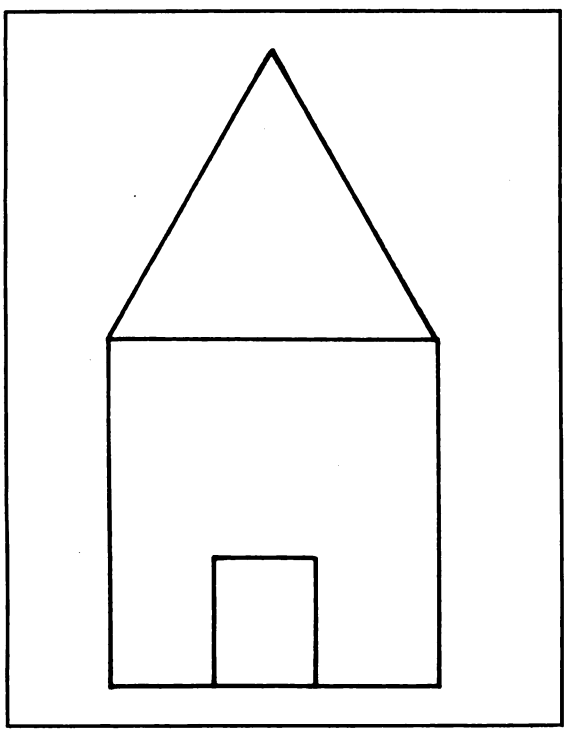
Really, there are two ways of drawing a picture. You can simply start by using the

keyboard to tell Sam to do things and experiment as you go along. If something doesn't turn out the way that you wanted it to, you simply try again OR you can use the editor to make changes to the procedure.

If you find that part of the way through the picture, Sam leads you to a different idea, then that is all right. This can be a very successful way of discovering some very interesting patterns. You can also discover a lot about Sam and a lot about how Logo and geometry works.

The second way to use Sam in drawing pictures is to plan the pictures in advance. By planning the pictures, you can make sure that you do not leave things out and that you get precisely what you want. It is a good idea to draw the picture on a piece of paper. When you have drawn it on the paper, you can decide how to break the problem down into smaller pieces.

You can also make complicated pictures out of smaller items. For example, if you have procedures that can draw houses and trees, you can combine them to make a bigger picture of a house with trees around it.



House.

Let's start with an example of drawing a house, and then we will be able to use the house procedure later. Here is the picture drawn on paper.

The house can be drawn by doing the following

Clearing the screen and HIDING the Turtle
Drawing a square
Moving the invisible turtle to the top left of the square
Drawing a triangle for the roof

Here is what you would need to type to do this

```
TO HOUSE
CS
HT
WALL
MOVE1
ROOF
END

TO MOVE1
FD 50
END

TO WALL
REPEAT 4 [FD 50 RT 90]
END

TO ROOF
RT 60
REPEAT 3 [FD 50 RT 120]
END
```

If you want to try something extra, you could try adding a door to the house. After you

have tried adding the door, you can try the solution shown below. Once again, your solution might be different from the one given here.

Here is the DOOR procedure written by itself.

```
TO DOOR
FD 30
LT 90
FD 20
LT 90
FD 30
END
```

The complete procedure for HOUSE will now look like this

```
TO HOUSE
HT
WALL
MOVE1
ROOF
MOVE2
DOOR
END

TO WALL
REPEAT 4 [FD 50 RT 90]
END

TO MOVE1
FD 50
END

TO ROOF
RT 30
REPEAT 3 [FD 50 RT 120]
END

TO MOVE2
RT 60
FD 50
RT 90
FD 50
RT 90
FD 10
RT 90
END

TO DOOR
FD 30
LT 90
FD 20
LT 90
FD 30
END
```

You could try adding some windows next. When you have a house that you like, you can try adding a tree, a cloud in the background, or a group of flowers. From the previous chapters, you will find procedures for most of these things. If you look back, then you will be able to use these procedures over again to make up a picture. When you have a picture or design that you like, you can then try to use color for the backgrounds and pen.

SETTING BACKGROUNDS AND COLORS

You will find that the use of color backgrounds and colors for the pen will add a lot to your drawings. You can change the background colors more than once during one drawing if you like. You should be sure that the pen colors and the background colors will look well together, and that you will not make your picture disappear by having a background color and a pen color with the same color code.

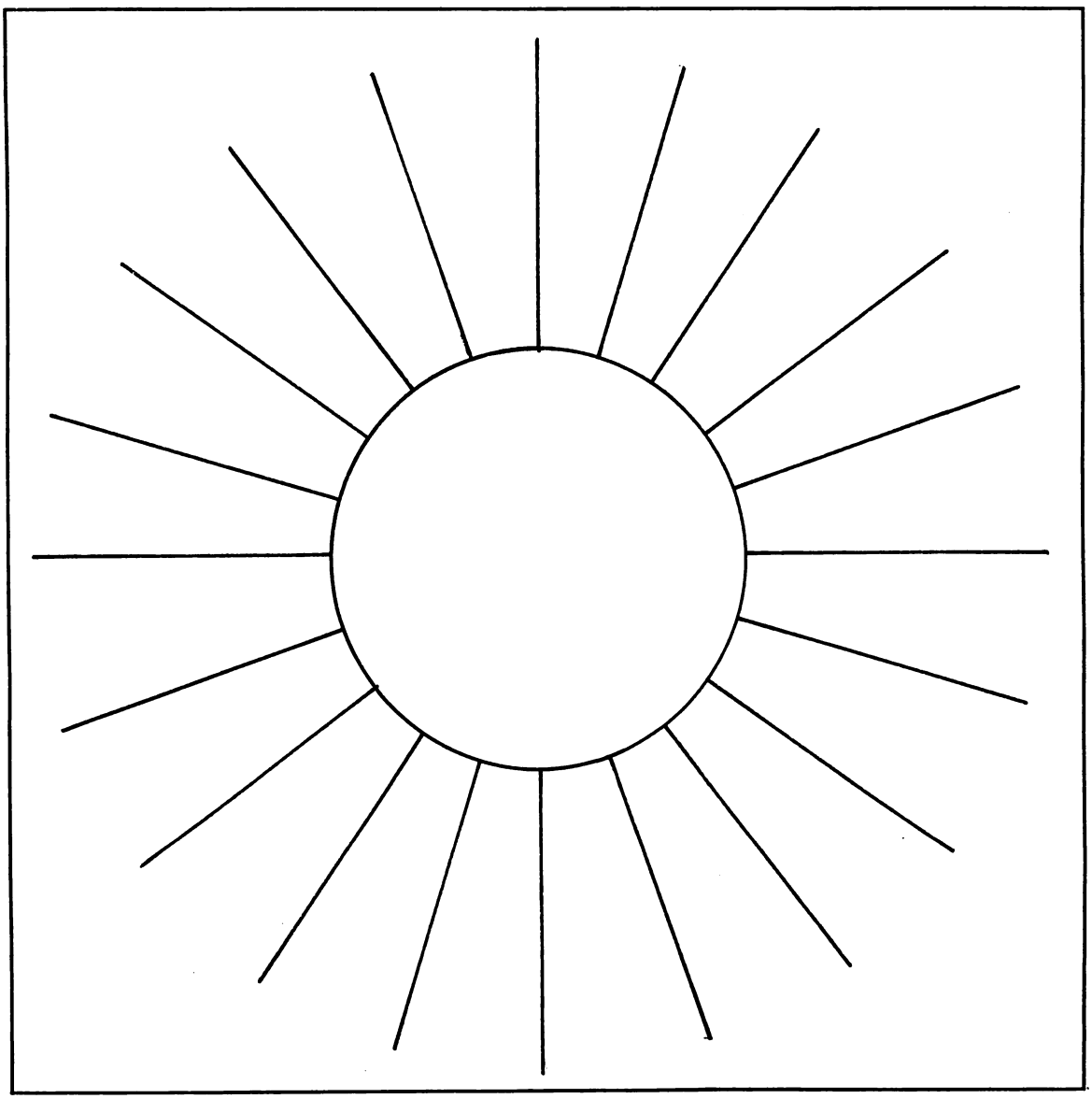
If you want to remove the turtle from the screen, include the primitive HT for HideTurtle at the beginning or at the very end of your procedure. Using HT will speed up the drawing process that Logo uses.

Here are some challenges for you to try. You can try them yourself before you look at

the given procedure, or you can look at the solution to give you an idea of how to get started. At the beginning of each challenge you will find an illustration of what you are trying to draw followed by a list of the steps needed, or should I say, the steps that will give one possible solution. You might find that your own steps will be different from the ones given here!

SUNSETS

Let's see if you can make a picture of a sunset. A sunset could be useful in later pictures. You could include a sunset in almost any



Sunset.

scene. Let's write an English list to begin with.

Here is the English list of statements that we might use.

Set the WINDOW
Draw a circle
Turn left 90
Draw a lot of rays

Draw a ray
Go forward a large distance
Go back the same distance
Turn right 90
Draw a little arc
Turn left 90

Here is the procedure for the sunset

```
TO SUN
HT
START
CIRCLER 50
LT 90
REPEAT 36 [RAY]
END

TO START
WINDOW
FULLSCREEN
SETBG 5
SETPC 4
END

TO RAY
FD 100
BK 100
RT 90
ARCR 50 10
LT 90
END
```

If you want to see the sunset, type SUN. Have you noticed that our watchdog always has a STOP sign with him? Let's see if we can get Sam to draw the stop sign.

"STOP RIGHT THERE!" SAID SAM

Let's have Sam draw a stop sign, the kind that you see on the highway and the kind that our watchdog carries! Page 178 shows what our stop sign might look like.

Let me see. . . how many sides do stop signs have again? If I remember correctly, they have 8 sides. Oh, and by the way, most of them have a border around them; that might be a challenge! Last, but not least, they *all* have the word STOP written on them.

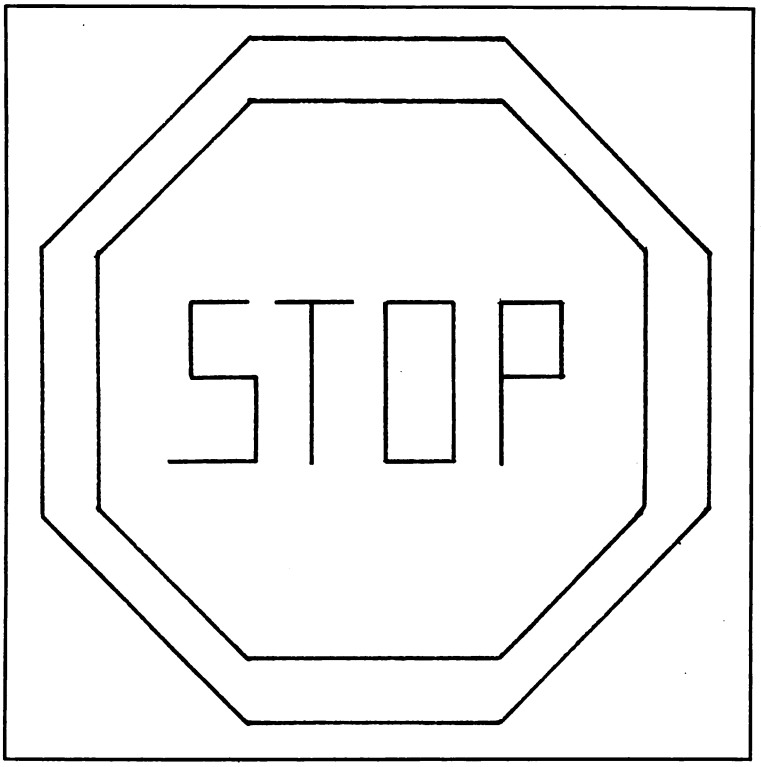
I think that we should draw the border first. Our English list of instructions for Sam would look like this.

Set Sam's position on the screen to a nice location
Draw the outside of the sign (8 sides)
Move inside the outline a little
Draw the inside border of the sign (8 sides)
Write the message STOP on the screen.
End

The outside and inside outlines are both octagons (That means eight sided as in OCTopus), so we should use one subprocedure for both of those. The subprocedure will need a variable, as we are going to have to change the size of the octagons.

Here is the main procedure that will control Sam's movements.

```
TO STOPSIGN
MAKE "SIZE 60
SET
DRAW.OCTAGON :SIZE
MOVE1
```



Stop sign.

```
DRAW.OCTAGON :SIZE-10
MOVE2
MESSAGE
HT
END
```

The MAKE primitive will tell Sam to put the value 60 into the container called SIZE. The quotation mark (") in front of the word SIZE indicates that SIZE is the label on the variable container (remember Sam's cookie jars?).

This is another way of introducing a variable into a procedure. With this method, you will not need to enter the value that you want as you call the procedure. The value will already be there. To change the value, you will have to change the number that follows the MAKE primitive.

The procedure MESSAGE will eventually write the word STOP on the screen. We didn't use the word STOP in the procedure though, as the word STOP already has a special meaning for Sam.

Here is the procedure for SET that will move Sam to a nice location for us.

```
TO SET
WINDOW
FULLSCREEN
SETBG 4
PU
SETPOS [-40 -40]
PD
END
```

Here is the first move procedure.

```
TO MOVE1
PU
SETPOS [-28 -34]
PD
END
```

This MOVE1 procedure does not contain any variables. It is written for this drawing alone. Because of this, it is not very elegant programming, but it will tell Sam to do the job for us in the meantime.

Here is the procedure to draw the octagons.

```
TO DRAW.OCTAGON :SIZE
REPEAT 8 [FD :SIZE RT 45]
END
```

Hmm. . . let me see, we have an 8 and a 45. What is 8 times 45? Why 360 of course, I wonder if that was a coincidence? This procedure is used twice in the program. The first time, it is called with a value of 60. The second time it is called with a value of 50, (:SIZE−10).

The second move procedure is called MOVE2. Here is what it looks like.

```
TO MOVE2
PU
SETPOS [−15 −20]
PD
END
```

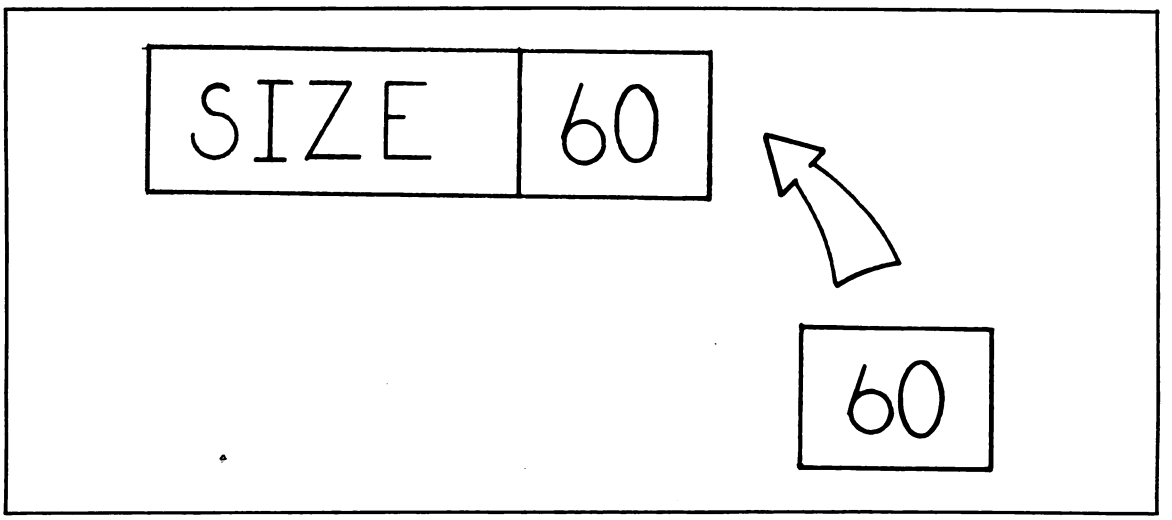
The MOVE2 procedure is also custom designed for the job. It has no variables, and cannot be adjusted easily for other jobs.

Finally, here is the MESSAGE subprocedure. We will break the subprocedure into smaller parts.

```
TO MESSAGE
S
T
O
P
END
```

Neat! Can you see the word STOP in the procedure? Here are the other procedures that you will need for the S, T, O, and P.

```
TO S
RT 90
FD 10
LT 90
FD 10
LT 90
FD 10
RT 90
```



Make "SIZE 60.

```
FD 10
RT 90
FD 10
END

TO T
PU
FD 10
PD
FD 20
BK 10
RT 90
FD 20
END

TO O
PU
LT 90
FD 20
PD
FD 20
LT 90
FD 20
LT 90
FD 20
LT 90
FD 20
END

TO P
PU
LT 90
FD 30
LT 90
PD
FD 20
RT 90
FD 20
RT 90
FD 10
RT 90
FD 20
END
```

Here is the complete procedure. If you want to see Sam draw the STOP sign, this is what you will need to type. Remember to use ERPS before you begin if it is needed.

```
TO STOPSIGN
MAKE "SIZE 60
SET
DRAW.OCTAGON :SIZE
MOVE1
DRAW.OCTAGON :SIZE-10
MOVE2
MESSAGE
HT
END

TO SET
WINDOW
FULLSCREEN
SETBG 4
PU
SETPOS [-40 -40]
PD
END

TO MOVE1
PU
SETPOS [-28 -34]
PD
END

TO DRAW.OCTAGON :SIZE
REPEAT 8 [FD :SIZE RT 45]
END

TO MOVE2
PU
SETPOS [-15 -20]
PD
END

TO MESSAGE
```

```
S
T
O
P
END

TO S
RT 90
FD 10
LT 90
FD 10
LT 90
FD 10
RT 90
FD 10
RT 90
FD 10
END

TO T
PU
FD 10
PD
FD 20
BK 10
RT 90
FD 20
END

TO O
PU
LT 90
FD 20
PD
FD 20
LT 90
FD 20
LT 90
FD 20
LT 90
FD 20
END

TO P
PU
LT 90
FD 30
LT 90
PD
FD 20
RT 90
FD 20
RT 90
FD 10
RT 90
FD 20
END
```

To see Sam draw the STOP sign, simply type the command STOP.

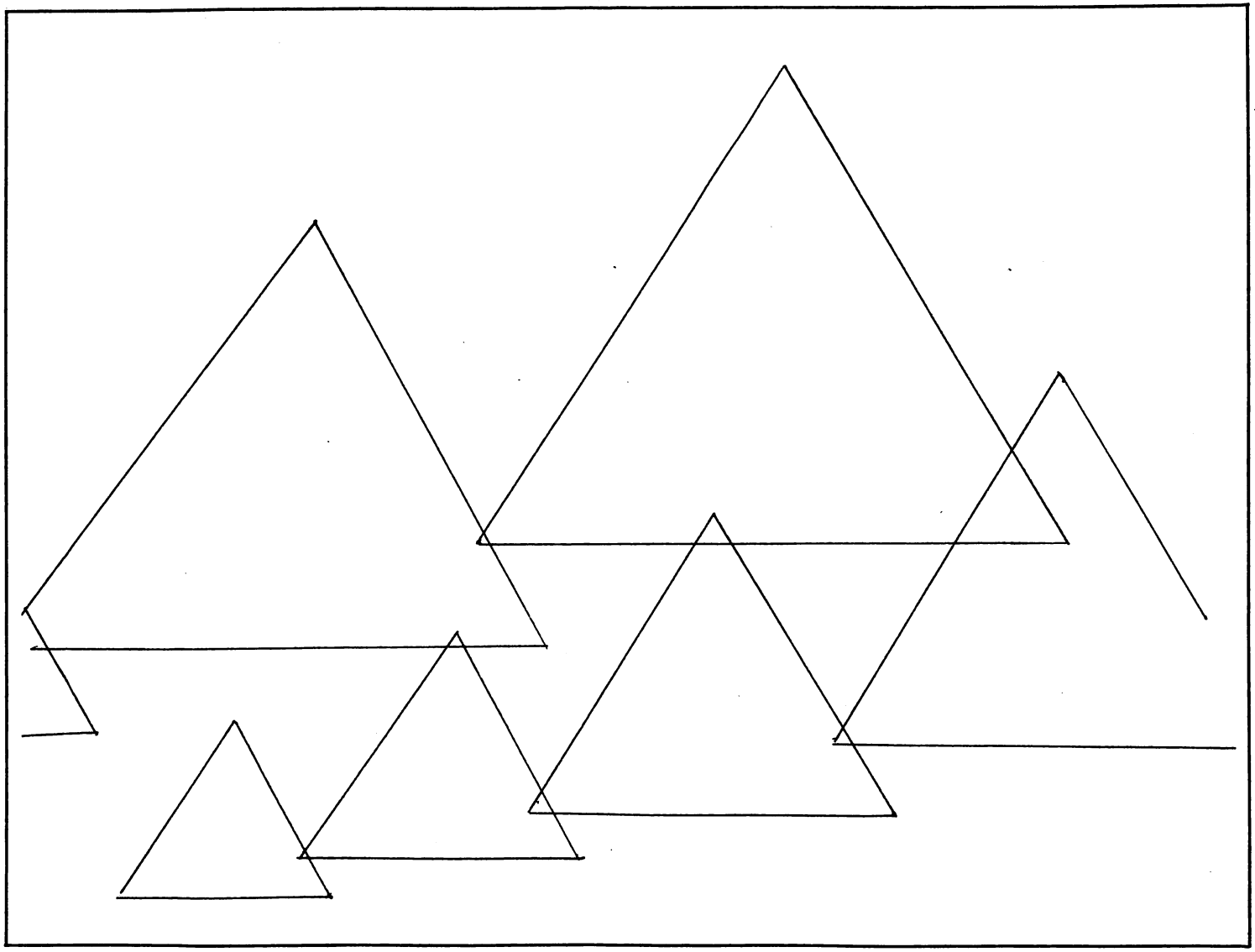
DRAWING MOUNTAINS

I wonder if we could draw some mountains? They would make a nice background for another picture. We could use them as a background for a house and maybe some trees!

Now let us see, mountains are . . . triangles, a lot of triangles drawn together. Have you ever looked at how mountains in the distance seem smaller than mountains that are close to you? We could make our mountains more realistic if we could make our triangles different sizes. That would mean using variables and probably recursion. The use of variables would allow us to adjust the size of the mountains, and the use of recursion would allow us to draw a lot of mountains easily. Recursion would also allow us to change the size of the variables to give us the effect of *perspective*. Perspective is the technique used when you make the distant objects seem smaller than nearby objects of the same size.

On the next page is an illustration of what we have in mind.

Here is our main procedure.



Mountains.

```
TO MOUNTAINS :SIZE
SET
MOUNTS :SIZE
END
```

We need the variable SIZE so that we can control the size of the mountains. The SET procedure will as usual start Sam off at an appropriate place on the screen. The procedure MOUNTS :SIZE is the procedure that will actually draw the mountains. It will have to be a recursive procedure that uses variables of course. Here is the procedure SET.

```
TO SET
FULLSCREEN
SETBG 2
PU
SETPOS [-120 -100]
RT 30
PD
END
```

Here is the recursive procedure MOUNTS :SIZE

```
TO MOUNTS :SIZE
IF :SIZE > 150 [HT STOP]
TRIANGLE :SIZE
MOVE :SIZE
MOUNTS :SIZE + 20
END
```

I thought that the procedure TRIANGLE would show up sooner or later! The recursive call in the procedure occurs just before the END statement. The procedure will be stopped by the IF condition if the mountains become too large. Finally, the MOVE procedure will simply move Sam over and up a bit so that he can begin the next mountain. Here is the MOVE procedure.

```
TO MOVE :SIZE
PU
RT 60
FD :SIZE * 0.8
LT 90
FD :SIZE * 0.2
RT 30
PD
END
```

Hmm. . . this MOVE is a little bit different! The RT 60 is unusual, I know! It has something to do with the fact that we have been drawing triangles. I bet that if we played turtle, it would help! The RT 60 is to bring Sam's heading back to due north.

Here is our famous triangle procedure once more.

```
TO TRIANGLE :SIZE
REPEAT 3[FD :SIZE RT 120]
END
```

You have seen this one many times, so that we will not discuss the details again. Here is the complete procedure for drawing mountains. This is what you would have to type to see Sam draw the picture. Remember to type ERPS, if it is needed before you begin.

```
TO MOUNTAINS :SIZE
SET
MOUNTS :SIZE
END

TO SET
FULLSCREEN
SETBG 2
PU
SETPOS [-120 -100]
RT 30
PD
END
```

```
TO MOUNTS :SIZE
IF :SIZE > 150 [HT STOP]
TRIANGLE :SIZE
MOVE :SIZE
MOUNTS :SIZE + 20
END

TO MOVE :SIZE
PU
RT 60
FD :SIZE * 0.8
LT 90
FD :SIZE * 0.2
RT 30
PD
END

TO TRIANGLE :SIZE
REPEAT 3 [FD :SIZE RT 120]
END
```

When you have typed in the procedure, you can watch Sam draw the mountains by typing MOUNTAINS 10. Try some other values in place of the 10 to see what will happen to the sizes of the mountains.

What else would look good with mountains? I know, trees would look good with mountains.

TREES

Here is a procedure that will allow us to draw a tree to go with the mountains. We will use variables and recursion in the procedure. We will not explain each step in the procedure. Can you read the procedure, and then write the English statements?

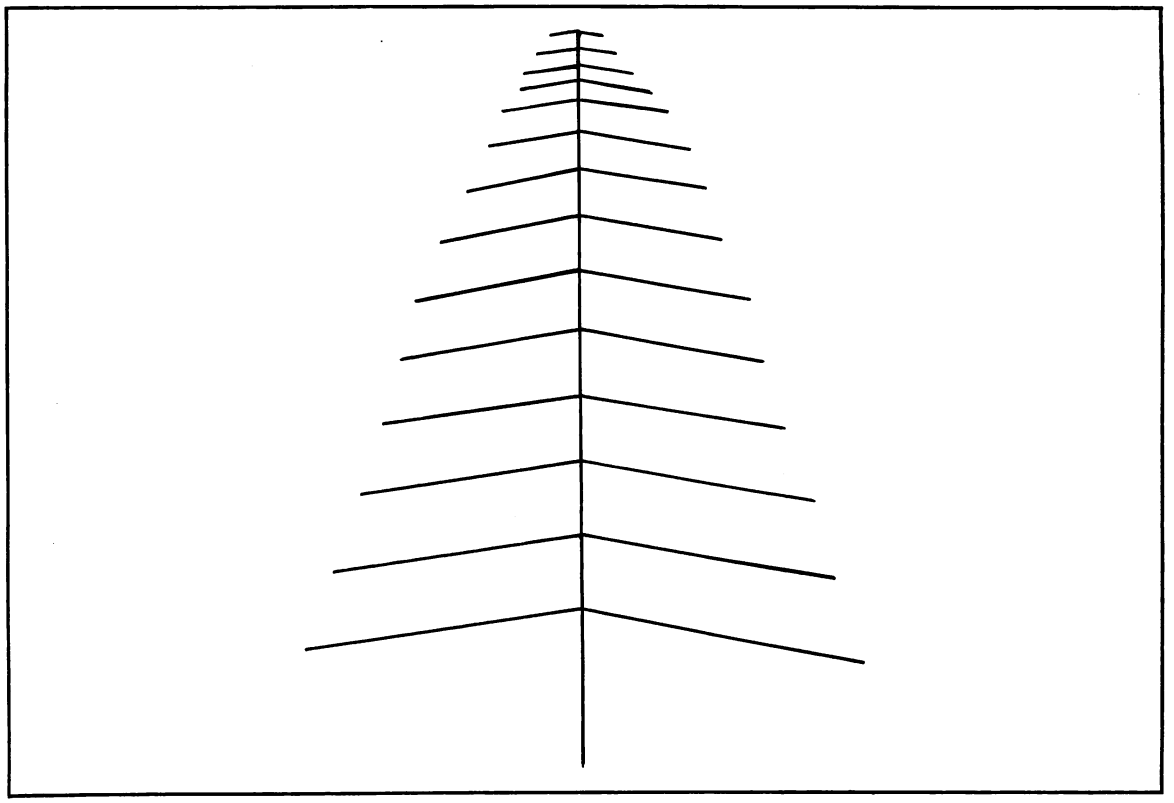
```
TO FIR.TREE :SIZE
SET
TRUNK :SIZE
BRANCH :SIZE
END

TO SET
WINDOW
FULLSCREEN
SETBG 2
PU
SETPOS [0 -120]
PD
END

TO TRUNK :SIZE
FD :SIZE
END

TO BRANCH:SIZE
IF :SIZE < 5 [STOP]
RT 100
LEAF
LT 200
LEAF
RT 100
FD :SIZE * 0.3
BRANCH :SIZE - 5
END

TO LEAF
FD :SIZE
BK :SIZE
END
```

To see Sam draw the tree, type FIR.TREE 60. You can use any value that you want for the procedure.

How small can you make the tree? Can you write a procedure that will draw a forest of trees?

PUTTING THE PARTS TOGETHER

We now have procedures that will draw mountains, houses and trees. It might be a good idea to draw a picture with all of these procedures in it. We could have the mountains as the main part of the picture, with a small



Tree.

house in the background and a tree near the house.

Here is a suggestion for the main procedure.

```
TO SCENE
MOUNTAINS
HOUSE
TREES
END
```

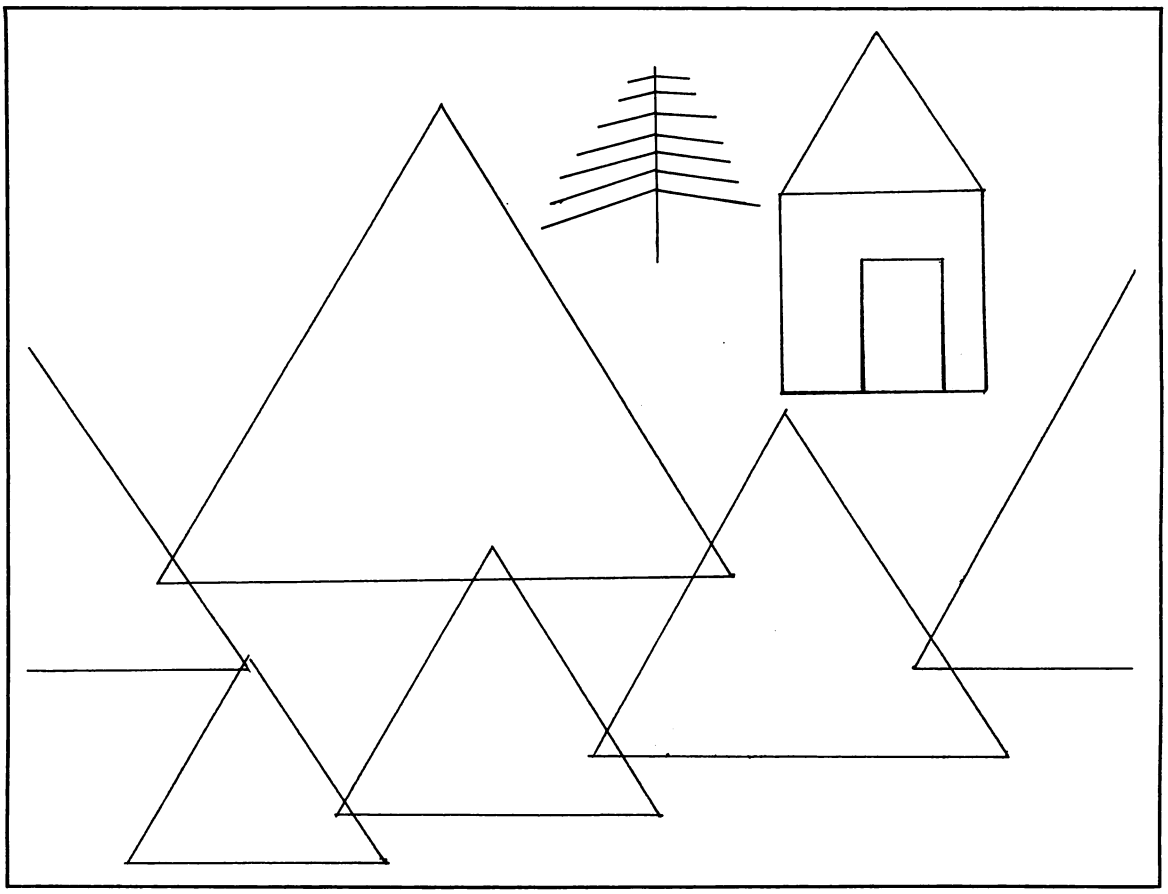
I think that we will have to include at least some variables so that we can make the mountains bigger than the house and so on. Here is the main procedure written with variables in it.

```
TO SCENE :SIZE
MOUNTAINS :SIZE * 3
POS1
HOUSE
POS2
FIR.TREE :SIZE * 2
END
```

The POS1 and the POS2 are simply procedures that will allow us to move Sam to new, suitable locations so that he can draw the next part of the scene for us.

The :SIZE * 3 and :SIZE * 2 should allow us to get the right proportions for the scene. If the proportions are still not quite correct, we can go back and change the values.

Here is the complete procedure. This is what you would have to type in to the machine to see the procedure draw a country scene. This must be one of the longest procedures so



Country scene.

far. It contains many procedures that you have already typed in. If you have those procedures on a disk, you might want to try loading them into Sam's memory, and then building this new procedure around the old ones.

```
TO SCENE :SIZE
MOUNTAINS :SIZE * 3
POS1
HOUSE
POS2
FIR.TREE :SIZE * 2
END

TO MOUNTAINS :SIZE
SET
MOUNTS :SIZE
END

TO SET
FULLSCREEN
SETBG 2
PU
SETPOS [-120 -100]
RT 30
PD
END

TO MOUNTS :SIZE
IF :SIZE > 150 [HT STOP]
```

```
TRIANGLE :SIZE
MOVE :SIZE
MOUNTS :SIZE + 20
END

TO TRIANGLE :SIZE
REPEAT 3 [FD :SIZE RT 120]
END

TO MOVE :SIZE
PU
RT 60
FD :SIZE * 0.8
LT 90
FD :SIZE * 0.2
RT 30
PD
END

TO POS1
PU
SETHEADING 0
SETPOS [40 20]
PD
END

TO HOUSE
HT
WALL
MOVE1
ROOF
MOVE2
END

TO WALL
REPEAT 4 [FD 50 RT 90]
END

TO MOVE1
FD 50
END

TO ROOF
RT 30
REPEAT 3 [FD 50 RT 120]
END

TO MOVE2
RT 60
FD 50
RT 90
FD 50
RT 90
FD 10
RT 90
END

TO DOOR
FD 30
LT 90
FD 20
LT 90
FD 30
END

TO POS2
PU
SETPOS [10 10]
SETHEADING 0
PD
END

TO FIR.TREE :SIZE
TRUNK :SIZE
BRANCH :SIZE
END

TO TRUNK :SIZE
FD :SIZE
END

TO BRANCH :SIZE
IF SIZE < 5 [STOP]
RT 100
LEAF :SIZE
LT 200
```

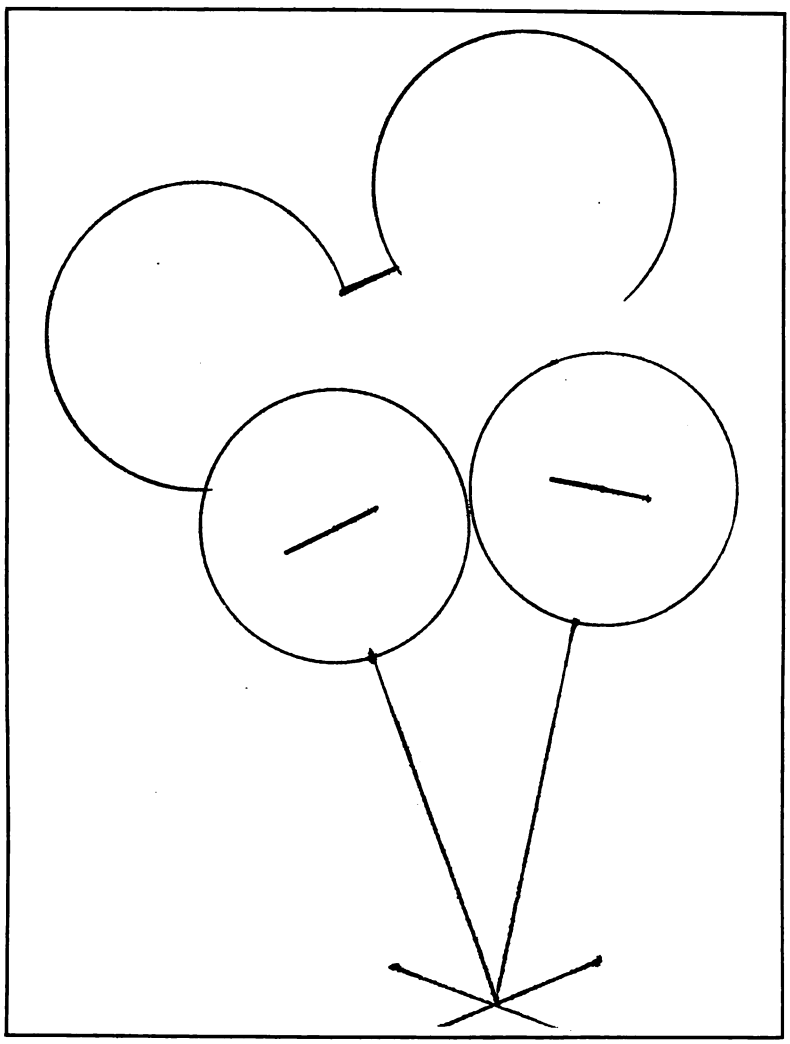
```
LEAF :SIZE
RT 100
FD :SIZE * 0.3
BRANCH :SIZE-5
END

TO LEAF :SIZE
FD :SIZE
BK :SIZE
END
```

When Sam was drawing the country scene, he made a new friend, a little friend with a squeaky voice. His name is Harry, and Harry is a mouse. Let's look at how to draw Harry.

HARRY MOUSE

Sam as you know likes lollipops. Did you know that you can do a lot of things with lollipops as well as drawing them and eating them? I wonder if we can draw a picture of Harry with some lollipops.

Here is a procedure that will draw our mouse using two lollipops. If you have saved



Mousey.

the procedure lollipop on disk, then you can load that procedure into Sam's memory and use it as a beginning for the mouse procedure.

```
TO MOUSE
HT
NOSE
WHISKERS
EYESL
EYESR
EARS
END

TO NOSE
RT 50
LOLLIPOP
HOME
RT 80
LOLLIPOP
HOME
END

TO LOLLIPOP
STICK
POP
END

TO STICK
FD 60
END

TO POP
LT 90
CIRCLER 20
END

TO WHISKERS
FD 20
BK 40
FD 20
LT 30
FD 20
BK 40
FD 20
END

TO EYESL
PU
HOME
RT 50
FD 80
LT 90
PD
FD 5
BK 10
PU
HOME
END

TO EYESR
PU
HOME
RT 80
FD 80
LT 90
PD
FD 5
BK 10
PU
HOME
END

TO EARS
MOVE
ARCR 21 260
LT 100
FD 10
LT 100
ARCR 21 260
END

TO MOVE
HOME
PU
RT 90
FD 65
LT 90
FD 73
```

```
LT 20
PD
END
```

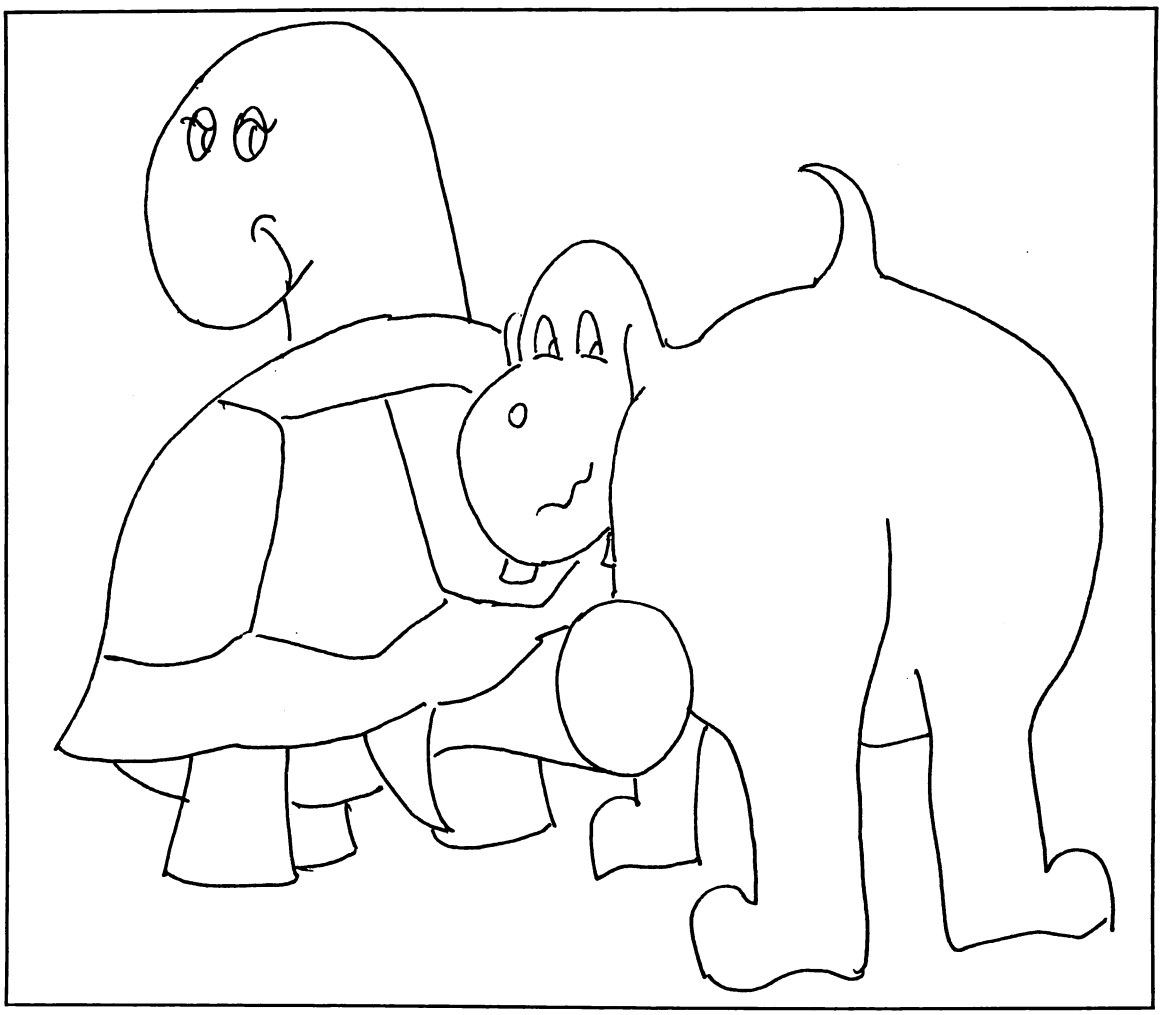
To see what Harry looks like, type MOUSE and then press the return key. Can you add some color to the procedure by inserting some color control primitives? How about changing the procedure so that the procedure will be called HARRY, or HARRY. MOUSE instead of just MOUSE.

Harry is quite big. Can you figure out how to draw a little Harry? If you can draw a little mouse, how about drawing a lot of mice on the screen at the same time.

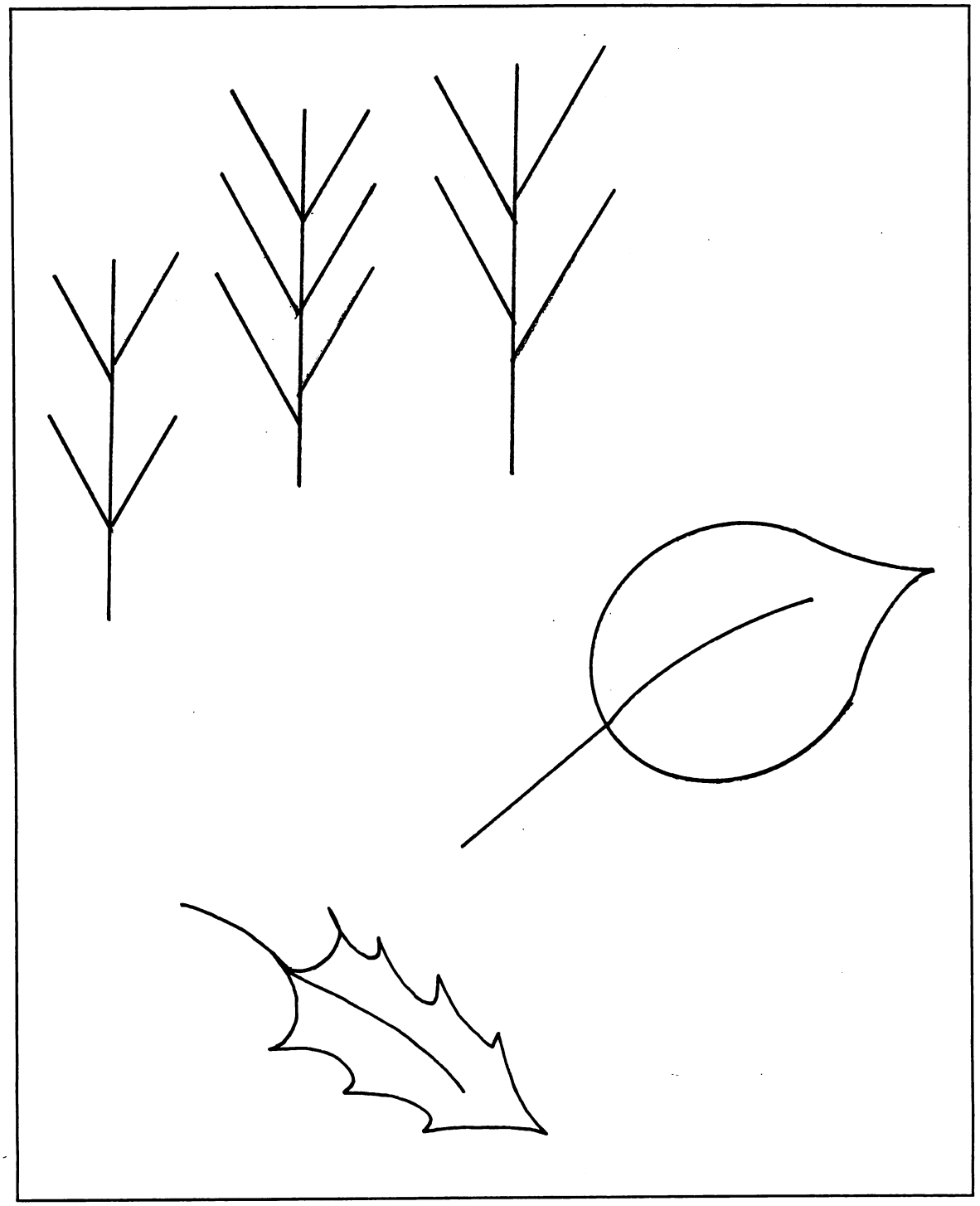
We are at the end of the last chapter. I hope that you have had fun with Sam and with Logo. Keep playing with the primitives and procedures. Good luck with Logo in the future!

CHALLENGES

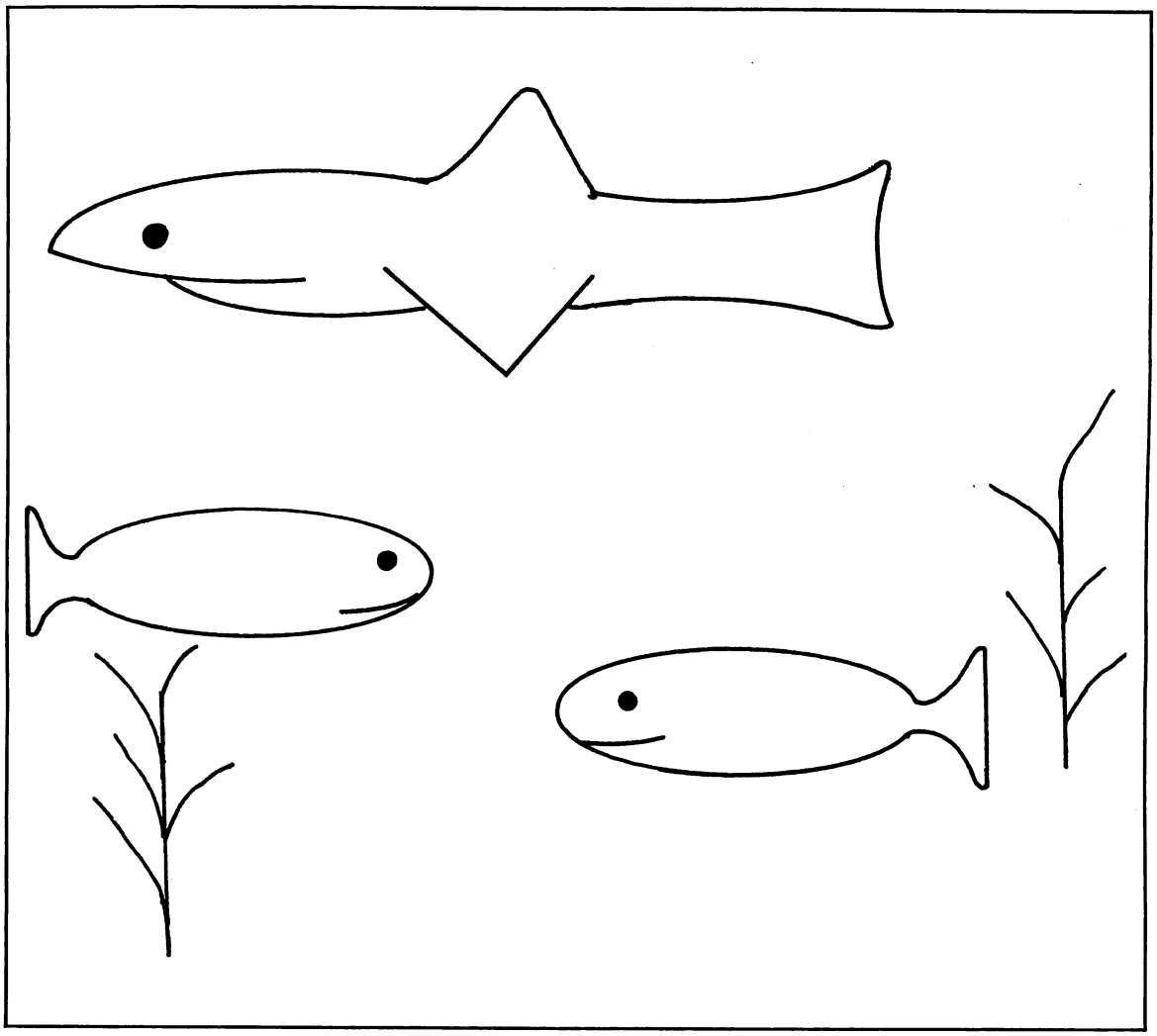
1. Have Sam draw a picture of a castle.



Sam and the watchdog say goodbye and good luck with Logo.



Autumn.



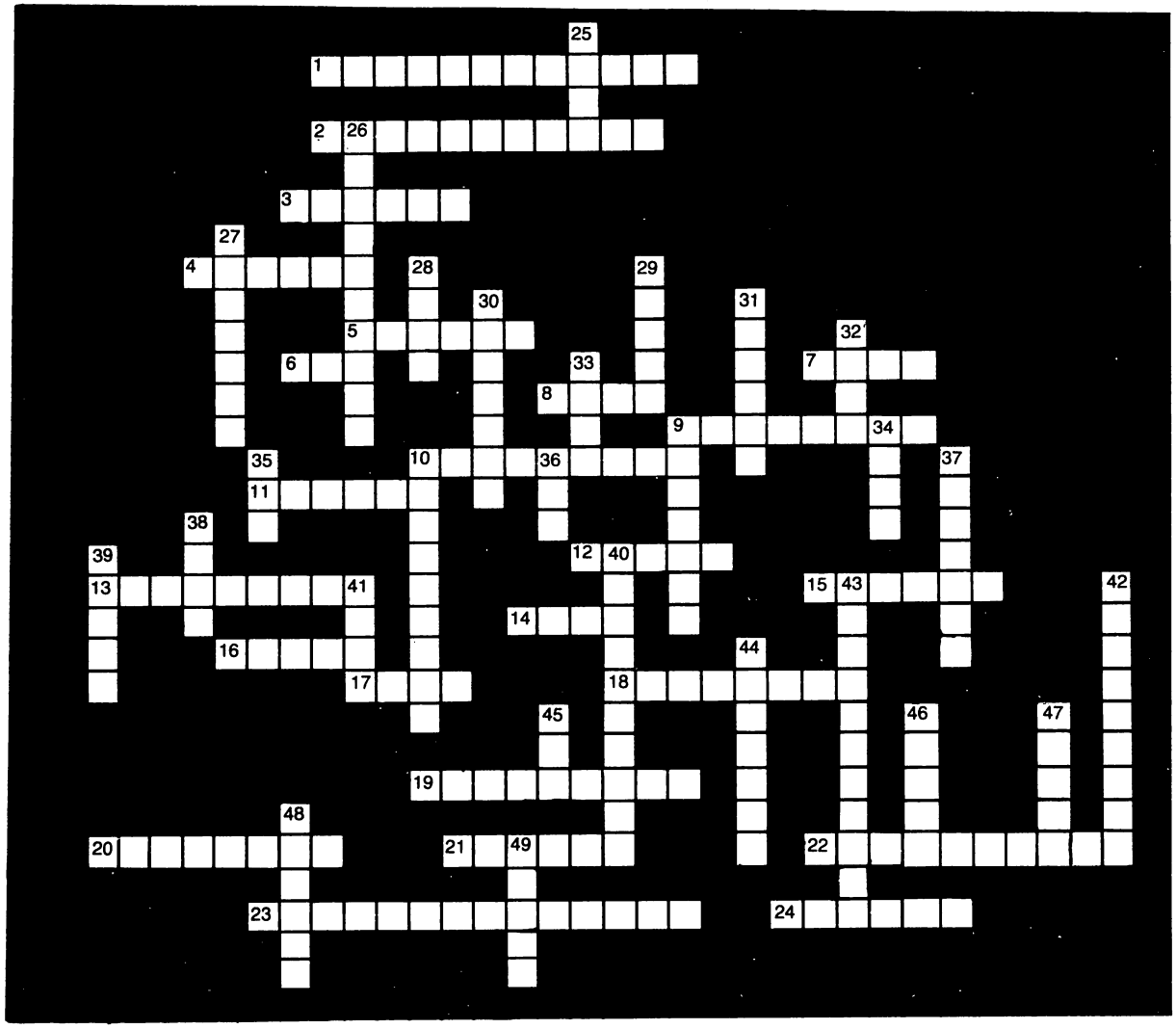
Ocean scene.

2. Surround the castle with trees.
3. Add clouds.
4. Add a sun in the sky.
5. Add a dragon in front of the castle.
6. Have Sam draw a picture of a tree with fallen leaves around it.
7. Have Sam draw a shark.
8. Add fish to the shark picture.
9. Add coral to the picture.
10. Tell Sam to write your initials on the bottom right corner of the picture.

CROSSWORD PUZZLE

Clues Across

1. When Sam is in an _____, he will do the same procedure forever and ever and ever.
2. When both Sam and some text are on the screen, you are in the _____ mode.
3. Number 4 is what you need here!
4. 12 divided by four should help!


25
1
2
26
3
27
4
28
29
30
31
5
32
6
33
7
8
9
34
35
10
36
37
11
38
39
12
40
13
41
15
43
42
14
16
44
17
18
45
46
47
19
48
20
21
49
22
23
24

5. Play it a few more times Sam!
6. ::::::::::—one of these could be called a _____.
7. "My native language", said Sam.
8. He starts here!
9. In order for him to learn a new procedure, we must _____ how to do it.
10. Lots of primitives make this up.
11. If you change your mind, use this!
12. If you change your mind, you can also use this one!
13. Type this if you are unhappy about your circle.
14. This one is a bit edgy!
15. This could help to stop you in style!
16. Sam I don't want you to draw this time!
17. To _____ a procedure is to send it to a disk.
18. The top of the heap is the _____.
19. This is what we are doing when we fix a procedure.
20. This is what our machine is.
21. Draw in a different color Sam!
22. You can read all of this.
23. We can change these.
24. He looks out of this.

Clues Down

25. (20 – 5)/3 is a pretty color.
26. Solve your problems this way!
27. This one will make Sam draw a circle to the right.
28. You could call your dad this!
29. To keep Sam inside, use this one.
30. I wonder what is on my disk?
31. I just have to remember the correct way to spell and punctuate.
32. You wash these after they have been used.
33. Get that from the disk, Sam!
9. Sam likes it when we do this to him!
34. This tells Sam to draw a fraction of a circle to the left.
35. Sam has one of these with a lot of colors in it.
10. String these together to make a procedure.
36. This is the last thing in your procedure.
37. Go left on this one.
38. Circle a little bit to the right on this one.
39. Change the paper color, Sam!
40. This is another word for "over and over again".
41. This will make things a little blank.
42. This means telling Sam to tell Sam to tell Sam . . . . . .
43. It is all yours to draw on.
44. It is time to draw again Sam!
45. This is a three letter word meaning a problem.
46. Use this to send Sam a number.
47. (30+2–9)/(3+20) will help our color.
48. A Disk is to Sam as this is to us.
49. This can be in a variable container.

Appendix A

A Parent's Guide to Logo

The parent's first question might be, "What is Logo?" The answer to that question is not necessarily a short one. Logo has many facets to its structure. First of all, Logo is a computer language that can be successfully used by children. Its very nature ensures that it will be successful with a wide variety of age groups and ability levels. Logo can easily challenge a college student, who is studying computing and artificial intelligence. At the same time, Logo can be used by first grade students. Logo, in short can be many things to many people. As a parent, you perhaps hope that it will introduce your children to computing. I have no hesitation in saying that it is the ideal introduction for your child. It is for instance, a much better introduction to computing than BASIC ever will be.

Logo will give your child a sound understanding of geometry and mathematics. The language is founded on the highest of educational philosophies. Logo comes from the most noble of parents. Logo has its roots in the theories of Piaget and in artificial intelligence languages. It was developed by Seymour Papert at M.I.T. It is the only computer language that was designed specifically for the educational field. It is perhaps one of the most powerful innovations of the decade!

WHAT CAN YOUR CHILD LEARN FROM LOGO?

What a child learns from Logo depends on a number of factors. The first factor is age. Logo can be used by the very young at a simple but rewarding level. It can also be used by graduate students studying robotics and animal behavior and be just as rewarding and just as challenging. The second factor is the child's interest. If he is not interested in computers and such things, he may not be interested in Logo. The third factor is *you*, the parent, who will be the childs first source of knowledge and

referral. For those of us who are not all that interested in computers and such, this might be difficult. Unfortunately, in today's world, a knowledge of computing is becoming mandatory, if only for the sake of our children.

The good news is that Logo is the easiest and by far the most pleasant way to become acquainted with computing. It is fun, and at the same time, it has enormous depth. Logo can be as complicated or simple as you want it to be! That is what makes it suitable for both children and university graduates.

Here is what your child can hope to gain from Logo and this book.

- Familiarity with computers and computer terms.
- Simple programming skills and discipline.
- Familiarity with basic computer functions including repetition and the use of procedures.
- Familiarity with a group of problem solving techniques.
- Familiarity with computer variables.
- Familiarity with the computer process called recursion.
- Lots of fun with the turtle and the computer, while learning about the above.

The last item is paradoxically the most important. Logo is designed to teach all of the above while being quite simply, FUN.

A Word of Warning

Logo was designed around a particular philosophy that espouses the idea that the student should find his own direction in his study of Logo. Logo is not intended to lead to right and wrong solutions. It is intended to lead the student to the things that interest him and to learning through the study of his mistakes. As a parent, you simply offer suggestions and help as they are needed. A brief summary of the philosophy of Logo is included in Appendix C.

BRIEFLY, HOW DOES LOGO WORK?

Initially, you will look at a small triangle on the screen called for historical reasons a *turtle*, or *Sam* as he is referred to in this book. You will very easily be able to tell the turtle to move forward (FD), right (RT), left (LT), and back (BK). The turtle can also be told to do other obvious items such as repeat things and change pen colors etc.

At the second level of operation, you can teach the turtle the meaning of words like HAT. The language proceeds in complexity to the level of a full fledged, sophisticated computer language of marked elegance.

WHAT CAN I DO TO HELP MY CHILD IN USING LOGO?

As a parent, the obvious questions are, "What should I know about Logo?" and "What constructive things can I do with Logo and my children?" The ideal answer to the first question is that a parent should learn the Logo language along with their children. This will prove to be easier than you might think. I know that I have had a tremendous amount of satisfaction playing with the computer and Logo with my own preschool children. Both my children and I have learned a great deal. If nothing else, it would be worth while for you and your child to simply have the exposure to the computer keyboard and computer terms, let alone the value of the exposure to a sophisticated but simple language like Logo.

Logo is simple, and it can be used at a variety of levels by a variety of age groups. It is simple to use, because the user can work on a simplified subset of the language. Begin at the beginning of the book, and work through the procedures carefully. Pick out the jargon words that are new to you and appears in

italics. Use these words in your conversations with your son or daughter as you discuss what you are doing together at the keyboard.

Do not be afraid to digress from the work at hand if a particular avenue seems to offer interest. Logo is designed for and supports this philosophy of discovery.

When you come to a suggested assignment or challenge, don't be afraid to offer judicious suggestions. What may seem obvious to you might not seem obvious to others. When this happens, you perhaps have a choice of two extremes. The first would be to give the child the answer (or a possible answer) immediately. The second would be to offer no assistance, but to trust that the child will eventually discover the solution to the problem by his own efforts. The wisest course is probably to strike a balance in the middle somewhere by offering a suggestion that is helpful. At the end of each chapter, you will find some useful questions, challenges, and crosswords. These are offered as an effective way of memorizing the material in the chapter.

YOUR HANDY DANDY LIST OF NEVER FAIL QUESTIONS!

Here is a list of phrases that have proven their worth over the years. When in doubt, ask one of these questions.

1. Where have you seen that idea before?
2. What is missing from the procedure?
3. Why is this item not quite correct?
4. What other procedures have we used like this one?
5. Could we break this problem down into smaller parts?
6. Let's play turtle (imitate the actions the turtle is to perform).
7. What would the turtle have to do next?
8. Let's work from the answer backwards!
9. Could there be a missing number that we need?
10. Could we need another primitive here?
11. What does that error message usually mean?
12. Can you find the page in the book/manual where this is mentioned?
13. How did we fix this problem last time?
14. What would happen if we . . . . . ?
15. Why don't we break this up into smaller procedures and then test each separate procedure?
16. Why not try that from the "top level"?
17. Lets look that up in the manual/index.

Of all of the techniques mentioned, one of the most important and powerful is the idea of "playing turtle". It is important because it is something concrete that the child can use from his experience to help in the solving of problems. It is very powerful. What might seem difficult in the abstractions of language will suddenly seem obvious in the concrete experience of playing turtle.

Playing Turtle

The child starts off by standing where the turtle started, except of course the child stands in the corresponding position in the middle of the room. The room, or a rug will correspond to the boundaries of the screen. The child then moves forward, right, back, etc. exactly as the turtle has done during the procedure. If it is a small child, the role of the adult is to take the child by the hand and to walk through the turtle trip with him. When the difficult parts are reached, the question is asked "What should we tell the turtle to do next?" You will be surprised at the lucidity of the answers that will be received! THE IMPORTANCE OF PLAYING AT TURTLE, ESPECIALLY WITH YOUNG CHILDREN CANNOT BE OVEREMPHASIZED. Use the idea a lot.

Telling Turtle

In telling turtle, you type single instruc-

tions or *primitives* into the computer. The computer will execute that instruction immediately. The idea of telling the *turtle*, as opposed to the *computer*, is very powerful! You are *telling turtle*, when you type in primitives such as FD 100 and RT 80.

Teaching Turtle

Not only can we *tell* the turtle what he is to do, we can also *teach* the turtle the meaning of procedures. If this is carefully phrased, it will appear as if the turtle is actually *learning* the meaning of the words. It is common in Logo to hear of children teaching turtle, this is often phrased as "teaching turtle to SQUARE" or "teaching turtle to HAT".

After the turtle is taught to SQUARE, he will respond to the command SQUARE by drawing a square (if that is what the procedure SQUARE actually specifies. There is no reason that you *have* to call the square SQUARE. You can in fact call a square MUGWUMP if that pleases you. If you do, then, when you type the command MUGWUMP, you will find that the turtle will draw a plain square once again.)

Appendix B

Getting Started with Apple Logo

Here is how you can make a fast start with your Apple Logo package. In the Apple package, you should find the following; a Logo tutorial, a Logo manual, and two Logo disks. Set the Logo manual to one side for the moment; the book that you will use initially will be the tutorial manual.

The two disks are identical. These are the Logo language disks. Both disks are copy protected. You will not be able to copy them. This is the reason for giving you two disks, one disk will act as a backup. Protect both these disks from damage by placing a write protect tab over each notch.

You need one more disk, which you will supply yourself. This disk will be used for storing your Logo procedures. Take a new, blank disk, and initialize it using your Apple System Master disk, the one that came with your Apple computer. Here is how to do this.

Turn on the Apple with the System Master disk in the drive. When the disk drive stops chattering, and the Apple is up and running, remove the system disk and insert the blank disk. Type the following:

```
INIT HELLO
```

The drive will whir and chatter, and the new disk will be initialized. This is the disk that you will use for your Logo files. Remove the new disk, and label it LOGO.FILES. Do not write protect the disk.

Take your Logo language disk and insert it into the disk drive. Switch off the machine, and then switch it on again. This will boot the Logo language system. The disk drive will whir and chatter once more, and the Logo language will load into the language card in the computer. After a second or two, the drive will stop, and you will receive a message on the screen that gives you a choice of pressing the

return key or changing disks. PRESS THE RETURN KEY *FIRST* IF YOU DO NOT DO THIS NOW, YOUR TURTLE WILL NOT KNOW HOW TO DRAW CIRCLES AND ARCS! The drive will run again, and some extra information will be loaded into the computer.

REMOVE THE LANGUAGE DISK NOW, AND REPLACE IT WITH THE FILE DISK THAT YOU JUST PREPARED. Use your language disk to start the system only. The language disk will last longer that way. You have only two copies of the language disk to use. You should now put these copies in a safe place. Use one as a working copy, and the other as a backup. If one fails, send it to the Apple company for a replacement.

To start with turtle graphics, type the letters ST. The letters stand for ShowTurtle. You are now in the graphics mode, and you are ready to use Logo.

Avoid using the RESET key while in Logo. If you do use the reset key, Logo will invite you to restart the system. If you are in doubt about what to do, you should take the Logo language disk and restart the system as outlined above.

Appendix C

Getting Started with Terrapin Logo

Here is how you can make a fast start with Terrapin Logo. In your package from Terrapin, you will find one manual in a ring binder, and two disks. One disk is a language disk, and it cannot be copied. You should arrange for the purchase of a backup disk immediately. The disk is write protected to guard against accidental damage.

The second disk is a utilities disk that can be copied. Copy this disk now. Use the COPYA program that came on the System Master disk that came with your Apple computer.

See page B-2 in your Terrapin manual to find out how to prepare a blank Logo disk for your procedures.

To start the system, insert your language disk, and switch the Apple on. The drive will chatter and whir, and the Logo language will be loaded into the language card in the computer. When the loading is complete, REMOVE THE LANGUAGE DISK, AND INSERT THE COPY OF YOUR LOGO UTILITIES DISK. This will make your copy of the language disk last longer! Put the language disk away in a safe place.

You are now ready to use Terrapin Logo. To enter the graphics mode, type the word DRAW in response to the prompt and cursor. From this point on, the turtle will obey commands, such as FD 100, RT 90 etc.

Appendix D

Summary of Common Apple Logo Commands

PRIMITIVES

FD 52	Forward 52 steps
BK 30	Back 30 steps
RT 120	Right 120 degrees
LT 40	Left 40 degrees
HT	Hide the turtle
ST	Show the turtle
CS	Clear screen
PD	Put pen down
PC	Pen color in use
SETPC 3	Set pencolor 3 (violet)
PU	Pick pen up
PENERASE	Erase the trace
PENREVERSE	Reverse the trace
SETBG 3	Set the background to violet
Colors	0-black, 1-white 2-green, 3-violet 4-orange, 5-blue
WAIT 60	Wait one second

PR HEADING	Display the turtle's heading
PR POSITION PR POS	Give the turtle's position

THE OUTSIDE WORLD

PR [HELLO])	Print a message on the video
READCHAR	Read a character from the keyboard
READLIST	Read a list from the keyboard

THE KEYBOARD

CTRL-G	Stop processing
[shift] M	Display closed brackets
[shift] N	Display open brackets

FILES

CATALOG	Display the catalog
LOAD "SQUARE	Load the file SQUARE
SAVE "SQUARE	Save the file SQUARE
ERASEFILE "SQUARE	Erase the file on disk called SQUARE

TEXT AND SCREEN COMMANDS

HOME	Send the turtle to his home position facing north
FENCE	Stop the turtle from moving off the screen
WINDOW	Eliminate wraparound
TEXTSCREEN	Use the whole screen for text

FULLSCREEN	Use the whole screen for the turtle
SPLITSCREEN	Gives the graphics mode with text on the bottom four lines
CLEARTEXT	Clear the textscreen
CTRL-L	Use the FULLSCREEN
CTRL-S	Use the SPLITSCREEN
CTRL-T	Use the TEXTSCREEN

EDITING

EDIT "SQUARE	Enter the edit mode and display the procedure SQUARE.
backspace arrow	Erase the character to the left of the cursor
CTRL-A	Move to the beginning of the edit line
CTRL-B	Do a backspace
CTRL-C	Exit the editor
CTRL-D	Erase the character under the cursor
CTRL-E	Move the cursor to the end of the edit line
CTRL-F	Move the cursor forward
CTRL-K	"Kill" and store the edit line in a buffer.
CTRL-N	Move to the next line
CTRL-O	Open a new edit line
CTRL-P	Move to the previous line
CTRL-Y	Insert the current line from the "Kill" buffer.
[return]	Move the cursor and the following text to the beginning of the next line

THE WORKSPACE

POTS	Print out the title lines of the procedures
POPS	Print out the procedures
PO "SQUARE	Print out the primitives in SQUARE
ERASE "SQUARE or ER "SQUARE	Erase SQUARE from memory
ERPS	Erase all procedures from the workspace.
STOP	Stop the execution of the procedure

VARIABLES

FD :SIDE	Move forward by the amount side
TO SQUARE :SIDE	Indicates that SQUARE will need an input for the value side; eg. SQUARE 10
PR 5+3	Print the result of the computation
TO POLY :SIDE :ANGLE	Indicates that two values must be input
MAKE "START 25	Assign the value 25 to the variable START

CIRCLES AND ARCS

CIRCLER 10	Display a circle to the right with a radius of 10
CIRCLEL 10	Display a circle to the left with a radius of 10
ARCR 30 144	Display an arc to the right with a radius of 30 and 144 degrees of arc

ARCL 30 144	Displays an arc to the left with a radius of 30 and 144 degrees of arc.

CONDITION AND FLOW OF CONTROL

REPEAT 5 [(SQUARE RT 45)]	Repeat the procedure SQUARE followed by RT 45 5 times
CTRL-W	Cause the computer to "wait". Press any key to pressed to continue.

Appendix E

Equivalent Commands and Keystrokes in Apple Logo, Terrapin Logo, TI Logo, and TRS-80 Color Logo

Text and Screen Commands

Function	APPLE LOGO	TERRAPIN LOGO	TI LOGO	RS LOGO
Turtle draw mode	ST	DRAW	TELL TURTLE	R
Clear screen	CS	CS	CS	CLEAR
Set turtle position	SETPOS [10 100]	SETXY 10 100	SXY 10 100?	SX 10 SY 100
Set pen color	SETPC 3	PENCOLOR 3	SC 3	PC 3
Set background	SETBG	BACKGROUND	CB	BG 4
Penreverse	PENREVERSE	PENCOLOR 6	PR	---
Penerase	PE	PENCOLOR 6	PE	PC background color
Window	WINDOW	NOWRAP	---	---
Fence	FENCE	NOWRAP	---	NOWRAP
Fullscreen graphics	CTRL-L	CTRL-F	---	---

Editing

Delete	left arrow	ESC key	clear, delete,	[SHIFT] back arrow
Move cursor left	CTRL-B	left arrow	left arrow	back arrow
Move cursor right	right arrow	right arrow	right arrow	right arrow
Insert line buffer	CTRL-Y	---	---	---
Edit a procedure	ED "BOX	ED BOX	EDIT BOX	E
Exit the editor	CTRL-C	CTRL-C	BACK	[BREAK]
Print a procedure	PO "BPX	PO BOX	PO BOX	

		Editing		
Erase procedures	ERPS	ER PROCEDURES	ERASE BOX	[SHIFT] [CLEAR]
Open, close brackets	CTRL-M, N	CTRL-M, N	SHIFT-4, 5	[ ]
Print out titles	POTS	POTS	PP	use the editor
Print out proc.	POPS	PO PROCEDURES	PA	P (uses printer)
		Files		
Catalog	CATALOG	CATALOG	see manual	see manual
Save to file	SAVE "BOX	SAVE "BOX	SAVE see manual	S see manual
Load from file	LOAD "BOX	READ "BOX	RECALL	L see manual
		Control Statements		
IF statements	IF :S > 5 [STOP]	IF :S > 5 STOP	IF :S > 5 THEN STOP	IF :S > 5 [STOP]

Appendix F

Crossword Puzzle Solutions

CHAPTER 1

Across

1. RIGHT
2. FORWARD
3. CLEARSCREEN
4. LOGO
5. PRIMITIVE
6. LEFT
7. WRAPAROUND

Down

8. HOMEPOSITION
9. BACKWARDS
10. TURTLE
3. COMPUTER
11. SAM

CHAPTER 2

Across

1. CATALOG
2. ERPS
3. TOSQUARE
4. DEFINED
5. REPETITION
6. SAVE

Down

7. HOME
8. TELLSAM
9. PROCEDURE
10. LOAD
11. END
12. TEACHSAM

CHAPTER 3

Across

1. WRITEALIST
2. TOPLEVEL
3. PLAYTURTLE
4. INTUITION
5. LOGOISFUN
6. SMALLSTEPS

Down

7. DIFFERENCES
8. DEBUGGING
2. TRIALANDERROR
9. SYNTAX
10. PROBLEM

CHAPTER 4

Across

1. [CTR]S
2. [CTRL]P
3. EDITOR
4. POPS
5. [CTRL]L
6. BACKARROW
7. [CTRL]O

Down

8. ED"SQUARE
9. [CTRL]C
1. [CTRL]T
10. POTS
11. [CTRL]B
12. [CTRL]A
3. ERASE
5. [CTRL]N

CHAPTER 5

Across

1. BLUE
2. SETBACKGROUND
3. ORANGE
4. WINDOW
5. WRAP
6. GREEN

Down

7. SPLITSCREEN
8. FULLSCREEN
9. FENCE
10. PENUP
11. PENDOWN
2. SETPENCOLOR
12. TEXTSCREEN
13. VIOLET

CHAPTER 6

Across

1. CIRCLER
2. DEGREESOFARC
3. DISK

Down

4. ARCL
5. ARCR
1. CIRCLEL
6. RADIUS

CHAPTER 7

Across

1. CONTAINER

2. INPUT
3. LOCALVARIABLES
4. NUMBER

Down

5. THING
6. VARIABLE
7. DOT
8. CALL
9. SQUARE65

CHAPTER 8

Across

1. ANYKEY
2. IFTHEN
3. CTRLG
4. RECURSION
5. REPETITION
6. [CTRL]W

Down

7. INFINITELOOP
8. CHASEHISTAIL

CHAPTER 9

Across

1. INFINITELOOP
2. SPLITSCREEN
3. ORANGE
4. VIOLET
5. REPEAT
6. DOT
7. LOGO
8. HOME
9. TEACHSAM
10. PROCEDURE
11. EDITOR
12. ERASE
13. ED"CIRCLE
14. WRAP
15. IFTHEN
16. PENUP
17. SAVE
18. TOPLEVEL
19. DEBUGGING
20. COMPUTER
21. SETPEN
22. TEXTSCREEN
23. LOCALVARIABLES
24. WINDOW

Down

25. BLUE
26. PLAYTURTLE
27. CIRCLER
28. POPS
29. FENCE
30. CATALOG
31. SYNTAX
32. POTS
33. LOAD
9. TELLSAM
34. ARCL
35. PEN
10. PRIMITIVE
36. END
37. CIRCLEL
38. ARCR
39. SETBG
40. REPETITION
41. ERPS
42. RECURSION
43. FULLSCREEN
44. PENDOWN
45. BUG
46. INPUT
47. WHITE
48. MEMORY
49. THING

Index

A

Arcs, 106

B

Background colors, 83
Brackets, 155
Bug, 4

C

Catalog, 26
Changing a variable, 130
Changing the display, 64
Changing the screen, 90
Circles, 103, 147
Clear screen, 5
Colors, 83, 175
Commands, 202
Commands, Terrapin Logo, 207
Common problem, 47
Control keys, 68
Crossword puzzle solutions, 209
Cursor, 1

D

Daisy chain, 167
Dandelion, 36
Debugging, 45
Decrement, 158
Defining procedures, 20
Degrees of arc, 107
Disks, 26
Displaying titles, 63
Dots, 122

E

Editor, 62
Egypt, 138
End, 20
Erasing characters, 72
Erasing memory, 26
Erasing procedures, 27, 62
Errors, 45

F

Fence, 94
Flag, 13
Flower, 19, 131
Forward, 1

G

Graphics, 67

H

Heading, 149
Home position, 4
How-to problems, 48

I

IF, 148
Increments, 154
Infinite loops, 145
Input, 122
Inserting lines, 73
Intuition, 144

K

Keys, control, 68

L

Language problems, 47
Left, 9
Levels, 48
Listing instructions, 54
Loading procedures, 28
Logo, background of, 195
Lollipops, 114
Loops, 145

M

Mountains, 181
Mouse, 188

P

Papert, Seymour, 195
Parents, 195
Pen color, 86
Pen down, 88
Pen up, 88
Permanent memory, 104
Perspective, 181
Piaget, theories of, 195
Pictures, 172
Planning a picture, 174
Playing turtle, 52, 197
Polygons, 30, 152
Primitive, 4
Printing a message, 151
Printing out procedures, 64
Problem solving, 44
Procedure, 19
Procedure definitions, 76
Programmable screens, 91
Prompt, 1
Pyramid, 138, 165

Q

Questions for parents, 197

R

Radius, 104
Recursion, 145
Renaming a procedure, 76
Repetition, 31, 33
Return key, 3
Right, 5
Rules, 55

S

Sam, introduction to, 1
Saving procedures, 26
Scene, 186
Setheading, 167
Solutions, crossword puzzle, 209
Special numbers, 55
Spiral, 131
Splitscreen, 67
Squares, 34
Starting Apple Logo, 199
Stop, 149
Stopping a procedure, 148
Stop sign, 177
Subprocedures, 134
Sunsets, 176
Syntax errors, 48

T

Teaching turtle, 198
Technical problems, 47
Telling turtle, 197
Temporary memory, 104
Terrapin logo, 201
Text-screens, 64
TI Logo, 207
Top-level, 48
Tower, 135, 163
Trace function, 57
Trees, 184
Triangle, 10, 29
TRS-80 Color Logo, 207

U

Umbrella, 116

V

Variables, 122
Variables, two, 129

W

Wait, 79, 93
Windmill, 36
Window, 93
Wraparound, 5, 93

Edited by Marilyn L. Johnson

Apple Logo for Kids

If you are intrigued with the possibilities of the programs included in *Apple Logo for Kids* (TAB BOOK No. 1728), you should definitely consider having the ready-to-run disk containing the software applications. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the disk eliminates the possibility of errors that can prevent the programs from functioning. Interested?

Available on disk for Apple II, II +, *//*e, 64K (requires Apple Logo disk) at $24.95 for each disk plus $1.00 each shipping and handling.

I'm interested. Send me:

_____ disk for Apple II, II +, *//*e, 64K (requires Apple Logo disk)

_____ TAB BOOKS catalog

_____ Check/Money Order enclosed for $ _____
plus $1.00 shipping and handling for each disk ordered.

_____ VISA _____ MasterCard

Account No. _____ Expires _____

Name _____

Address _____

City _____ State _____ Zip _____

Signature _____

Mail To: **TAB BOOKS INC.**
P.O. Box 40
Blue Ridge Summit, PA 17214

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)

TAB 1728

Apple Logo for Kids

by David A. Yule

Let "Sam" the turtle teach your children (or you) the painless way to learn the language of Logo. You'll be delighted at how quickly they pick up both basic and advanced techniques with this easy-to-understand guide!

And best of all, they'll have fun learning! They'll be amazed to find that they control Sam and can tell him what to do. They'll discover how to make him go forward and back, right or left. And, they must teach Sam the meaning of words, thus learning the words themselves. In no time at all, they'll have mastered activities such as erasing the contents of the computer's memory and saving procedures in disk files.

With Sam, they can color, draw circles and arcs, change the sizes of pictures and diagrams, and then put it all together to create sunsets, mountains, trees, and much more! Without even knowing it (it's our secret), they'll have learned math and logical reasoning as well as the basics of computer programming.

Although based on Apple Logo, this unique programming manual can be easily adapted for TI Logo, Terrapin Logo, and Radio Shack Color Logo by using a handy table to determine equivalent keystrokes.

And to ensure understanding, there's a helpful "watchdog" throughout the text wherever something is introduced that is tricky, used frequently, or especially helpful. At the end of each chapter your child will find summaries to reinforce important ideas, questions to test their understanding, crossword puzzles using new words, and programs and ideas for experimenting.

Entertaining as well as educational, this unique programming manual is the ideal way to learn Logo . . . a language that is simple enough for a child to master . . . yet complete enough to lay a solid programming foundation.

David Yule is a Professional Development Associate in Microcomputing with the British Columbia Teachers Association. He has taught grades 6 through college level and has been using Logo with students of many ages.

OTHER POPULAR TAB BOOKS OF INTEREST

Apple® Lisa: A User-Friendly Handbook (No. 1691—$16.50 paper; $22.95 hard)

Dragon Magic: Your First Programming Book (No. 1747—$8.25 paper only)

TAB TAB BOOKS Inc.

Blue Ridge Summit, Pa. 17214

Send for FREE TAB Catalog describing over 750 current titles in print.

FPT > $11.50

ISBN 0-8306-1728-0

PRICES HIGHER IN CANADA

1095-0584